**Forecasting Using Multiple Models**

**Matt Dancho**

**2019-10-08**

* [Prerequisites](https://cran.rstudio.com/web/packages/sweep/vignettes/SW02_Forecasting_Multiple_Models.html#prerequisites)
* [Forecasting Gasoline Prices](https://cran.rstudio.com/web/packages/sweep/vignettes/SW02_Forecasting_Multiple_Models.html#forecasting-gasoline-prices)
* [Performing Forecasts Using Multiple Models](https://cran.rstudio.com/web/packages/sweep/vignettes/SW02_Forecasting_Multiple_Models.html#performing-forecasts-using-multiple-models)
  + [Multiple Models Concept](https://cran.rstudio.com/web/packages/sweep/vignettes/SW02_Forecasting_Multiple_Models.html#multiple-models-concept)
  + [Multiple Model Implementation](https://cran.rstudio.com/web/packages/sweep/vignettes/SW02_Forecasting_Multiple_Models.html#multiple-model-implementation)
* [Inspecting the Model Fit](https://cran.rstudio.com/web/packages/sweep/vignettes/SW02_Forecasting_Multiple_Models.html#inspecting-the-model-fit)
  + [sw\_tidy](https://cran.rstudio.com/web/packages/sweep/vignettes/SW02_Forecasting_Multiple_Models.html#sw_tidy)
  + [sw\_glance](https://cran.rstudio.com/web/packages/sweep/vignettes/SW02_Forecasting_Multiple_Models.html#sw_glance)
  + [sw\_augment](https://cran.rstudio.com/web/packages/sweep/vignettes/SW02_Forecasting_Multiple_Models.html#sw_augment)
* [Forecasting the model](https://cran.rstudio.com/web/packages/sweep/vignettes/SW02_Forecasting_Multiple_Models.html#forecasting-the-model)
* [Tidying the forecast](https://cran.rstudio.com/web/packages/sweep/vignettes/SW02_Forecasting_Multiple_Models.html#tidying-the-forecast)
* [Recap](https://cran.rstudio.com/web/packages/sweep/vignettes/SW02_Forecasting_Multiple_Models.html#recap)

Extending broom to time series forecasting

One of the most powerful benefits of sweep is that it helps forecasting at scale within the “tidyverse”. There are two common situations:

1. Applying a model to groups of time series
2. Applying multiple models to a time series

In this vignette we’ll review how sweep can help the **second situation**: *Applying multiple models to a time series*.

**Prerequisites**

Before we get started, load the following packages.

**library**(tidyverse)

**library**(tidyquant)

**library**(timetk)

**library**(sweep)

**library**(forecast)

**Forecasting Gasoline Prices**

To start, let’s get some data from the FRED data base using tidyquant. We’ll use tq\_get() to retrieve the Gasoline Prices from 1990 through today (2019-10-08).

gas\_prices\_monthly\_raw <- **tq\_get**(

x = "GASREGCOVM",

get = "economic.data",

from = "1990-01-01",

to = "2016-12-31")

gas\_prices\_monthly\_raw

## # A tibble: 316 x 2

## date price

## <date> <dbl>

## 1 1990-09-01 1.26

## 2 1990-10-01 1.34

## 3 1990-11-01 1.32

## 4 1990-12-01 NA

## 5 1991-01-01 NA

## 6 1991-02-01 1.09

## 7 1991-03-01 1.04

## 8 1991-04-01 1.08

## 9 1991-05-01 1.13

## 10 1991-06-01 1.13

## # … with 306 more rows

Upon a brief inspection, the data contains 2 NA values that will need to be dealt with.

**summary**(gas\_prices\_monthly\_raw$price)

## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's

## 0.900 1.138 1.615 1.974 2.697 4.002 2

We can use the fill() from the tidyr package to help deal with these data. We first fill down and then fill up to use the previous and then post days prices to fill in the missing data.

gas\_prices\_monthly <- gas\_prices\_monthly\_raw %>%

**fill**(price, .direction = "down") %>%

**fill**(price, .direction = "up")

We can now visualize the data.

gas\_prices\_monthly %>%

**ggplot**(**aes**(x = date, y = price)) +

**geom\_line**(color = **palette\_light**()[[1]]) +

**labs**(title = "Gasoline Prices, Monthly", x = "", y = "USD") +

**scale\_y\_continuous**(labels = scales::dollar) +

**theme\_tq**()



Monthly periodicity might be a bit granular for model fitting. We can easily switch periodicity to quarterly using tq\_transmute() from the tidyquant package along with the periodicity aggregation function to.period from the xts package. We’ll convert the date to yearqtr class which is regularized.

gas\_prices\_quarterly <- gas\_prices\_monthly %>%

**tq\_transmute**(mutate\_fun = to.period, period = "quarters")

gas\_prices\_quarterly

## # A tibble: 106 x 2

## date price

## <date> <dbl>

## 1 1990-09-01 1.26

## 2 1990-12-01 1.32

## 3 1991-03-01 1.04

## 4 1991-06-01 1.13

## 5 1991-09-01 1.11

## 6 1991-12-01 1.08

## 7 1992-03-01 1.01

## 8 1992-06-01 1.14

## 9 1992-09-01 1.12

## 10 1992-12-01 1.08

## # … with 96 more rows

Another quick visualization to show the reduction in granularity.

gas\_prices\_quarterly %>%

**ggplot**(**aes**(x = date, y = price)) +

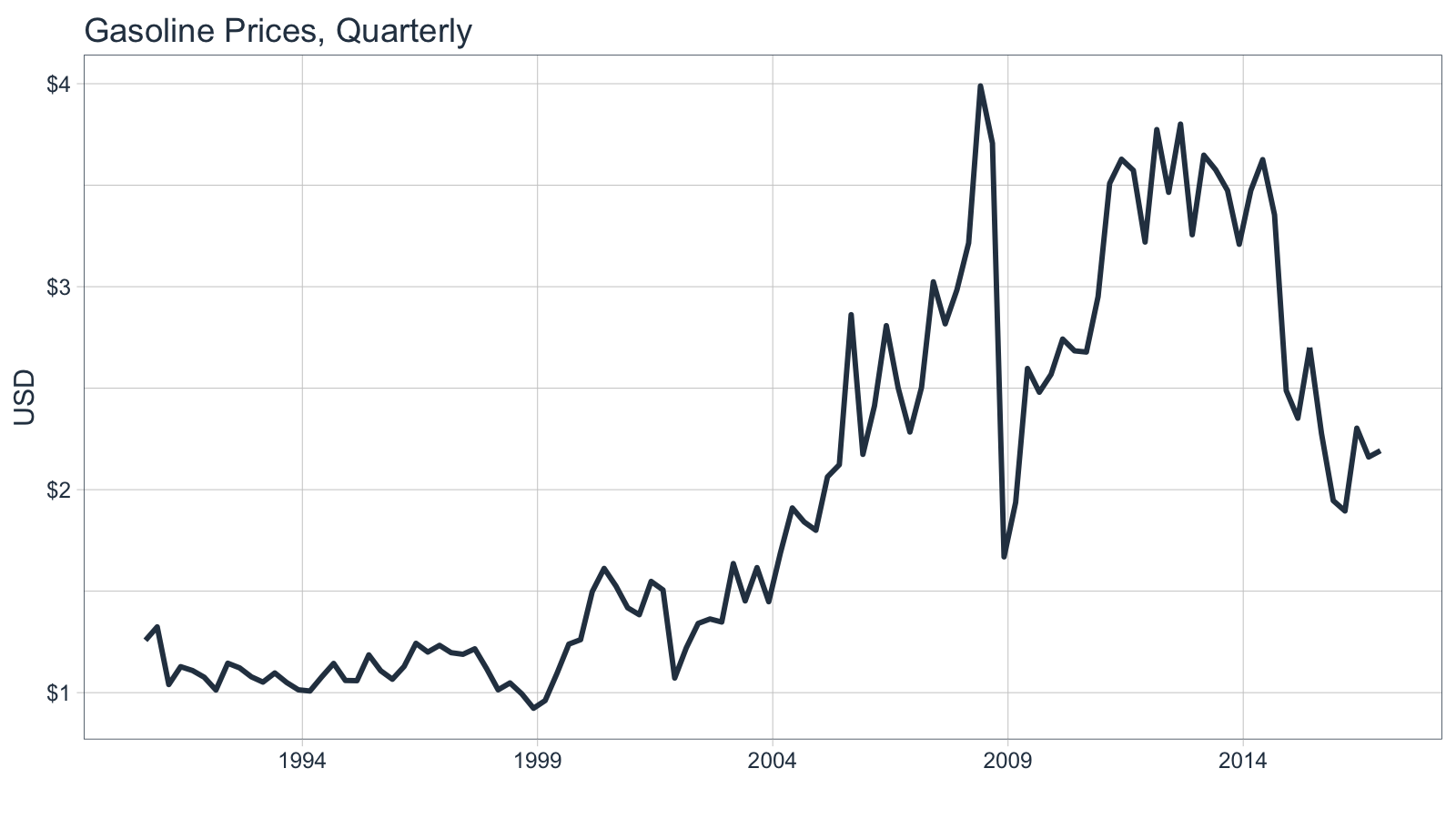
**geom\_line**(color = **palette\_light**()[[1]], size = 1) +

**labs**(title = "Gasoline Prices, Quarterly", x = "", y = "USD") +

**scale\_y\_continuous**(labels = scales::dollar) +

**scale\_x\_date**(date\_breaks = "5 years", date\_labels = "%Y") +

**theme\_tq**()



**Performing Forecasts Using Multiple Models**

In this section we will use three models to forecast gasoline prices:

1. ARIMA
2. ETS
3. BATS

**Multiple Models Concept**

Before we jump into modeling, let’s take a look at the multiple model process from [R for Data Science, Chapter 25 Many Models](http://r4ds.had.co.nz/many-models.html#from-a-named-list). We first create a data frame from a named list. The example below has two columns: “f” the functions as text, and “params” a nested list of parameters we will pass to the respective function in column “f”.

df <- **tibble**(

f = **c**("runif", "rpois", "rnorm"),

params = **list**(

**list**(n = 10),

**list**(n = 5, lambda = 10),

**list**(n = 10, mean = -3, sd = 10)

)

)

df

## # A tibble: 3 x 2

## f params

## <chr> <list>

## 1 runif <named list [1]>

## 2 rpois <named list [2]>

## 3 rnorm <named list [3]>

We can also view the contents of the df$params column to understand the underlying structure. Notice that there are three primary levels and then secondary levels containing the name-value pairs of parameters. This format is important.

df$params

## [[1]]

## [[1]]$n

## [1] 10

##

##

## [[2]]

## [[2]]$n

## [1] 5

##

## [[2]]$lambda

## [1] 10

##

##

## [[3]]

## [[3]]$n

## [1] 10

##

## [[3]]$mean

## [1] -3

##

## [[3]]$sd

## [1] 10

Next we apply the functions to the parameters using a special function, invoke\_map(). The parameter lists in the “params” column are passed to the function in the “f” column. The output is in a nested list-column named “out”.

df\_out <- df %>%

**mutate**(out = **invoke\_map**(f, params))

df\_out

## # A tibble: 3 x 3

## f params out

## <chr> <list> <list>

## 1 runif <named list [1]> <dbl [10]>

## 2 rpois <named list [2]> <int [5]>

## 3 rnorm <named list [3]> <dbl [10]>

And, here’s the contents of “out”, which is the result of mapping a list of functions to a list of parameters. Pretty powerful!

df\_out$out

## [[1]]

## [1] 0.3539214 0.8305328 0.1725492 0.3336279 0.4272752 0.1765578 0.8405683

## [8] 0.1642450 0.8351444 0.5650580

##

## [[2]]

## [1] 11 5 9 14 7

##

## [[3]]

## [1] -7.814870 -15.960203 -4.229270 -5.531148 -17.662205 -3.518611

## [7] 5.652761 5.394733 1.527318 13.632480

Take a minute to understand the conceptual process of the invoke\_map function and specifically the parameter setup. Once you are comfortable, we can move on to model implementation.

**Multiple Model Implementation**

We’ll need to take the following steps to in an actual forecast model implementation:

1. Coerce the data to time series
2. Build a model list using nested lists
3. Create the the model data frame
4. Invoke a function map

This is easier than it sounds. Let’s start by coercing the univariate time series with tk\_ts().

gas\_prices\_quarterly\_ts <- gas\_prices\_quarterly %>%

**tk\_ts**(select = -date, start = **c**(1990, 3), freq = 4)

gas\_prices\_quarterly\_ts

## Qtr1 Qtr2 Qtr3 Qtr4

## 1990 1.258 1.324

## 1991 1.040 1.128 1.109 1.076

## 1992 1.013 1.145 1.122 1.078

## 1993 1.052 1.097 1.050 1.014

## 1994 1.008 1.078 1.144 1.060

## 1995 1.059 1.186 1.108 1.066

## 1996 1.129 1.243 1.200 1.233

## 1997 1.197 1.189 1.216 1.119

## 1998 1.014 1.048 0.994 0.923

## 1999 0.961 1.095 1.239 1.261

## 2000 1.498 1.612 1.525 1.418

## 2001 1.384 1.548 1.506 1.072

## 2002 1.221 1.341 1.363 1.348

## 2003 1.636 1.452 1.616 1.448

## 2004 1.689 1.910 1.841 1.800

## 2005 2.063 2.123 2.862 2.174

## 2006 2.413 2.808 2.501 2.284

## 2007 2.503 3.024 2.817 2.984

## 2008 3.215 3.989 3.709 1.669

## 2009 1.937 2.597 2.480 2.568

## 2010 2.742 2.684 2.678 2.951

## 2011 3.509 3.628 3.573 3.220

## 2012 3.774 3.465 3.801 3.256

## 2013 3.648 3.576 3.474 3.209

## 2014 3.474 3.626 3.354 2.488

## 2015 2.352 2.700 2.275 1.946

## 2016 1.895 2.303 2.161 2.192

Next, create a nested list using the function names as the first-level keys (this is important as you’ll see in the next step). Pass the model parameters as name-value pairs in the second level.

models\_list <- **list**(

auto.arima = **list**(

y = gas\_prices\_quarterly\_ts

),

ets = **list**(

y = gas\_prices\_quarterly\_ts,

damped = TRUE

),

bats = **list**(

y = gas\_prices\_quarterly\_ts

)

)

Now, convert to a data frame using the function, enframe() that turns lists into tibbles. Set the arguments name = "f" and value = "params". In doing so we get a bonus: the model names are the now convieniently located in column “f”.

models\_tbl <- **enframe**(models\_list, name = "f", value = "params")

models\_tbl

## # A tibble: 3 x 2

## f params

## <chr> <list>

## 1 auto.arima <named list [1]>

## 2 ets <named list [2]>

## 3 bats <named list [1]>

We are ready to invoke the map. Combine mutate() with invoke\_map() as follows. Bada bing, bada boom, we now have models fitted using the parameters we defined previously.

models\_tbl\_fit <- models\_tbl %>%

**mutate**(fit = **invoke\_map**(f, params))

models\_tbl\_fit

## # A tibble: 3 x 3

## f params fit

## <chr> <list> <list>

## 1 auto.arima <named list [1]> <fr\_ARIMA>

## 2 ets <named list [2]> <ets>

## 3 bats <named list [1]> <bats>

**Inspecting the Model Fit**

It’s a good point to review and understand the model output. We can review the model parameters, accuracy measurements, and the residuals using sw\_tidy(), sw\_glance(), and sw\_augment().

**sw\_tidy**

The tidying function returns the model parameters and estimates. We use the combination of mutate and map to iteratively apply the sw\_tidy() function as a new column named “tidy”. Then we unnest and spread to review the terms by model function.

models\_tbl\_fit %>%

**mutate**(tidy = **map**(fit, sw\_tidy)) %>%

**unnest**(tidy) %>%

**spread**(key = f, value = estimate)

## # A tibble: 20 x 6

## params fit term auto.arima bats ets

## <list> <list> <chr> <dbl> <dbl> <dbl>

## 1 <named list [1… <fr\_ARIMA> ar1 0.834 NA NA

## 2 <named list [1… <fr\_ARIMA> ma1 -0.964 NA NA

## 3 <named list [1… <fr\_ARIMA> sar1 0.939 NA NA

## 4 <named list [1… <fr\_ARIMA> sma1 -0.776 NA NA

## 5 <named list [1… <bats> alpha NA 5.88e-1 NA

## 6 <named list [1… <bats> ar.coefficients NA NA NA

## 7 <named list [1… <bats> beta NA NA NA

## 8 <named list [1… <bats> damping.parame… NA NA NA

## 9 <named list [1… <bats> gamma.values NA -2.62e-2 NA

## 10 <named list [1… <bats> lambda NA 6.05e-5 NA

## 11 <named list [1… <bats> ma.coefficients NA 2.56e-1 NA

## 12 <named list [2… <ets> alpha NA NA 8.31e-1

## 13 <named list [2… <ets> b NA NA -5.24e-2

## 14 <named list [2… <ets> beta NA NA 1.00e-4

## 15 <named list [2… <ets> gamma NA NA 5.21e-2

## 16 <named list [2… <ets> l NA NA 1.29e+0

## 17 <named list [2… <ets> phi NA NA 8.37e-1

## 18 <named list [2… <ets> s0 NA NA 4.69e-2

## 19 <named list [2… <ets> s1 NA NA -2.09e-2

## 20 <named list [2… <ets> s2 NA NA -4.07e-2

**sw\_glance**

Glance is one of the most powerful tools because it yields the model accuracies enabling direct comparisons between the fit of each model. We use the same process for used for tidy, except theres no need to spread to perform the comparison. We can see that the ARIMA model has the lowest AIC by far.

models\_tbl\_fit %>%

**mutate**(glance = **map**(fit, sw\_glance)) %>%

**unnest**(glance, .drop = TRUE)

## Warning: The `.drop` argument of `unnest()` is deprecated as of tidyr 1.0.0.

## All list-columns are now preserved.

## This warning is displayed once per session.

## Call `lifecycle::last\_warnings()` to see where this warning was generated.

## # A tibble: 3 x 15

## f params fit model.desc sigma logLik AIC BIC ME RMSE MAE

## <chr> <list> <lis> <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>

## 1 auto… <name… <fr\_… ARIMA(1,1… 0.298 -20.6 51.2 64.4 0.0180 0.291 0.177

## 2 ets <name… <ets> ETS(M,Ad,… 0.118 -76.6 173. 200. 0.0149 0.292 0.170

## 3 bats <name… <bat… BATS(0, {… 0.116 159. 179. 184. 0.0193 0.259 0.159

## # … with 4 more variables: MPE <dbl>, MAPE <dbl>, MASE <dbl>, ACF1 <dbl>

**sw\_augment**

We can augment the models to get the residuals following the same procedure. We can pipe (%>%) the results right into ggplot() for plotting. Notice the ARIMA model has the largest residuals especially as the model index increases whereas the bats model has relatively low residuals.

models\_tbl\_fit %>%

**mutate**(augment = **map**(fit, sw\_augment, rename\_index = "date")) %>%

**unnest**(augment) %>%

**ggplot**(**aes**(x = date, y = .resid, group = f)) +

**geom\_line**(color = **palette\_light**()[[2]]) +

**geom\_point**(color = **palette\_light**()[[1]]) +

**geom\_smooth**(method = "loess") +

**facet\_wrap**(~ f, nrow = 3) +

**labs**(title = "Residuals Plot") +

**theme\_tq**()

## Don't know how to automatically pick scale for object of type yearqtr. Defaulting to continuous.



**Forecasting the model**

Creating the forecast for the models is accomplished by mapping the forecast function. The next six quarters are forecasted withe the argument h = 6.

models\_tbl\_fcast <- models\_tbl\_fit %>%

**mutate**(fcast = **map**(fit, forecast, h = 6))

models\_tbl\_fcast

## # A tibble: 3 x 4

## f params fit fcast

## <chr> <list> <list> <list>

## 1 auto.arima <named list [1]> <fr\_ARIMA> <forecast>

## 2 ets <named list [2]> <ets> <forecast>

## 3 bats <named list [1]> <bats> <forecast>

**Tidying the forecast**

Next, we map sw\_sweep, which coerces the forecast into the “tidy” tibble format. We set fitted = FALSE to remove the model fitted values from the output. We set timetk\_idx = TRUE to use dates instead of numeric values for the index.

models\_tbl\_fcast\_tidy <- models\_tbl\_fcast %>%

**mutate**(sweep = **map**(fcast, sw\_sweep, fitted = FALSE, timetk\_idx = TRUE, rename\_index = "date"))

models\_tbl\_fcast\_tidy

## # A tibble: 3 x 5

## f params fit fcast sweep

## <chr> <list> <list> <list> <list>

## 1 auto.arima <named list [1]> <fr\_ARIMA> <forecast> <tibble [112 × 7]>

## 2 ets <named list [2]> <ets> <forecast> <tibble [112 × 7]>

## 3 bats <named list [1]> <bats> <forecast> <tibble [112 × 7]>

We can unnest the “sweep” column to get the results of all three models.

models\_tbl\_fcast\_tidy %>%

**unnest**(sweep)

## # A tibble: 336 x 11

## f params fit fcast date key price lo.80 lo.95 hi.80 hi.95

## <chr> <list> <lis> <lis> <date> <chr> <dbl> <dbl> <dbl> <dbl> <dbl>

## 1 auto.… <name… <fr\_… <for… 1990-09-01 actu… 1.26 NA NA NA NA

## 2 auto.… <name… <fr\_… <for… 1990-12-01 actu… 1.32 NA NA NA NA

## 3 auto.… <name… <fr\_… <for… 1991-03-01 actu… 1.04 NA NA NA NA

## 4 auto.… <name… <fr\_… <for… 1991-06-01 actu… 1.13 NA NA NA NA

## 5 auto.… <name… <fr\_… <for… 1991-09-01 actu… 1.11 NA NA NA NA

## 6 auto.… <name… <fr\_… <for… 1991-12-01 actu… 1.08 NA NA NA NA

## 7 auto.… <name… <fr\_… <for… 1992-03-01 actu… 1.01 NA NA NA NA

## 8 auto.… <name… <fr\_… <for… 1992-06-01 actu… 1.14 NA NA NA NA

## 9 auto.… <name… <fr\_… <for… 1992-09-01 actu… 1.12 NA NA NA NA

## 10 auto.… <name… <fr\_… <for… 1992-12-01 actu… 1.08 NA NA NA NA

## # … with 326 more rows

Finally, we can plot the forecasts by unnesting the “sweep” column and piping to ggplot().

models\_tbl\_fcast\_tidy %>%

**unnest**(sweep) %>%

**ggplot**(**aes**(x = date, y = price, color = key, group = f)) +

**geom\_ribbon**(**aes**(ymin = lo.95, ymax = hi.95),

fill = "#D5DBFF", color = NA, size = 0) +

**geom\_ribbon**(**aes**(ymin = lo.80, ymax = hi.80, fill = key),

fill = "#596DD5", color = NA, size = 0, alpha = 0.8) +

**geom\_line**(size = 1) +

**facet\_wrap**(~f, nrow = 3) +

**labs**(title = "Gasoline Price Forecasts",

subtitle = "Forecasting multiple models with sweep: ARIMA, BATS, ETS",

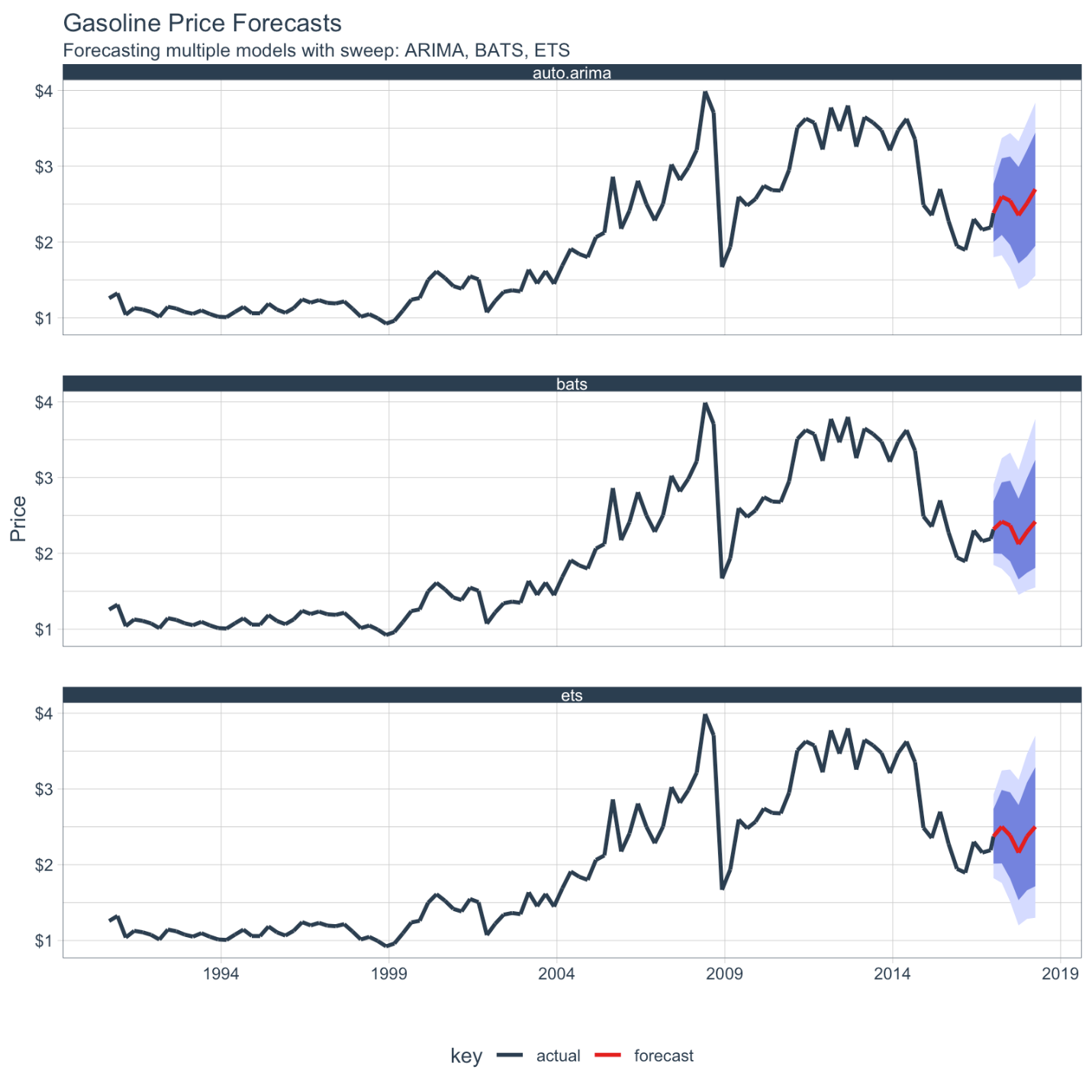
x = "", y = "Price") +

**scale\_y\_continuous**(labels = scales::dollar) +

**scale\_x\_date**(date\_breaks = "5 years", date\_labels = "%Y") +

**theme\_tq**() +

**scale\_color\_tq**()



**Recap**

The sweep package can aid analysis of multiple forecast models. In the next vignette we will review time series object coercion with sweep.