Nested Forecasting Approach and modeltime Methods

Dr. Kam Tin Seong
Assoc. Professor of Information Systems

School of Computing and Information Systems,
Singapore Management University

2022-7-16 (updated: 2022-07-28)

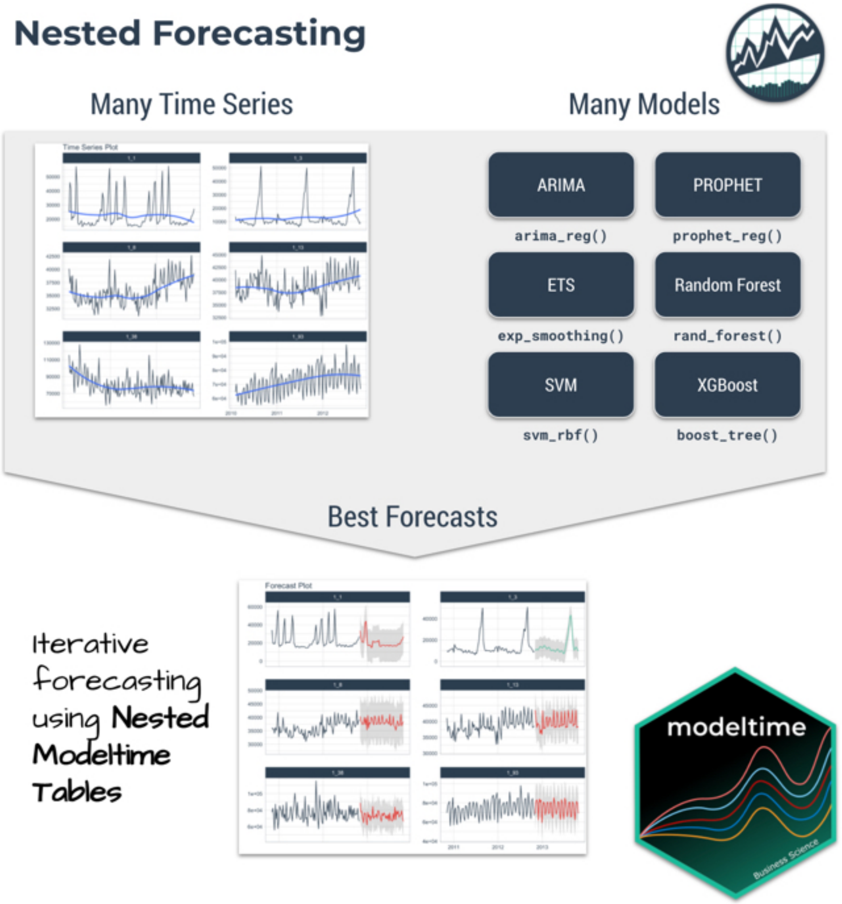
Content

- The motivation
- Principles of nested forecasting approach
- Nested forecasting processes
- Nested forecasting with modeltime

Motivation of Nested Forecasting Approach

In real world practice, it is very common a forecaster is required to forecast multiple time series by fitting multiple models.

Nested Forecasting



Source: Getting Started with Modeltime

Setting Up R Environment

For the purpose of this hands-on exercise, the following R packages will be used.

```
pacman::p_load(tidyverse, tidymodels,
               timetk, modeltime)
```

- **tidyverse** provides a collection of commonly used functions for importing, wrangling and visualising data. In this hands-on exercise the main packages used are readr, dplyr, tidyr and ggplot2.
- **modeltime** a new time series forecasting package designed to speed up model evaluation, selection, and forecasting. modeltime does this by integrating the **tidymodels** machine learning ecosystem of packages into a streamlined workflow for tidyverse forecasting.

The data

In this sharing, **Store Sales - Time Series Forecasting: Use machine learning to predict grocery sales** from Kaggle competition will be used. For the purpose of this sharing, the main data set used is *train.csv*. It consists of six columns. They are:

- *id* contains unique id for each records.
- *date* gives the sales dates.
- *store_nbr* identifies the store at which the products are sold.
- *family* identifies the type of product sold.
- *sales* gives the total sales for a product family at a particular store at a given date. Fractional values are possible since products can be sold in fractional units (1.5 kg of cheese, for instance, as opposed to 1 bag of chips).
- *onpromotion* gives the total number of items in a product family that were being promoted at a store at a given date.

For the purpose of this sharing, I will focus of grocery sales instead of all products. Code chunk below is used to extract grocery sales from *train.csv* and saved the output into an rds file format for subsequent used.

```
grocery <- read_csv(
  "data/store_sales/train.csv") %>%
  filter(family == "GROCERY") %>%
  write_rds(
    "data/store_sales/grocery.rds")
```

Step 1: Data Import and Wrangling

In the code chunk below, `read_rds()` of **readr** package is used to import grocery.rds data into R environment. Then, `mutate()`, `across()` and `as.factor()` are used to convert all values in columns 1,3 and 4 into factor data type.

```
grocery <- read_rds(
  "data/store_sales/grocery.rds") %>%
  mutate(across(c(1, 3, 4),
                as.factor)) %>%
  filter(date >= "2015-01-01")
```

In the code chunk below, `read_csv()` is used to import *stores.csv* file into R environment. TThen, `mutate()`, `across()` and `as.factor()` are used to convert values in columns 1to 5 into factor data type.

```
stores <- read_csv(
  "data/store_sales/stores.csv") %>%
  mutate(across(c(1:5),
                as.factor)) %>%
  select(store_nbr, cluster)
```

Data integration and wrangling

In the code chunk below, `left_join()` of **dplyr** package is used to join *grocery* and *stores* tibble data frames by using *store_nbr* as unique field.

```
grocery_stores <- left_join(
  x = grocery,
  y = stores,
  by = "store_nbr")
```

In the code chunk below, a new tibble data frame called *grocery_cluster* is derived by summing sales values by values in cluster and date fields.

```
grocery_cluster <- grocery_stores %>%
  group_by(cluster, date) %>%
  summarise(value = sum(sales)) %>%
  select(cluster, date, value) %>%
  set_names(c("id", "date", "value")) %>%
  ungroup()
```

Visualising the time series data: The code chunk

It is always a good practice to visualise the time series graphically.

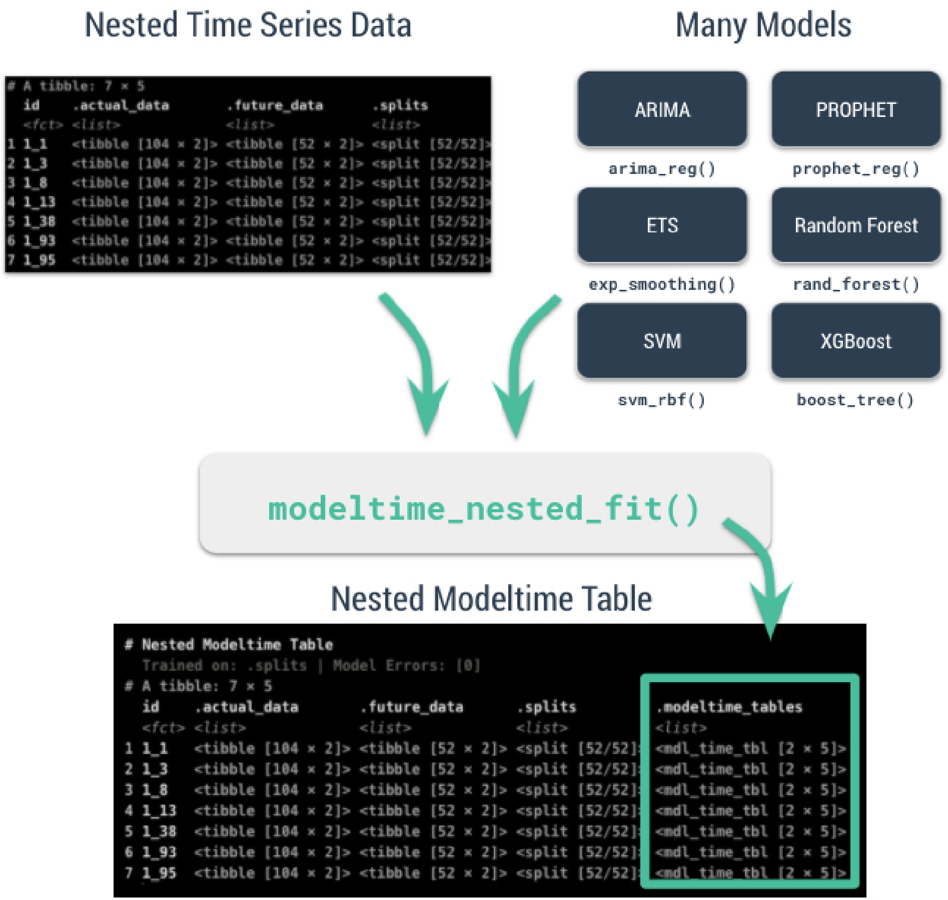
```
grocery_cluster %>%
  group_by(id) %>%
  plot_time_series(
    date, value,
    .line_size = 0.4,
    .facet_ncol = 5,
    .facet_scales = "free_y",
    .interactive = FALSE,
    .smooth_size = 0.4)
```

Visualising the time series data: The plot

Preparation for Nested Forecasting

Before fitting the nested forecasting models, there are two key components that we need to prepare for:

- **Nested Data Structure:** Most critical to ensure your data is prepared (covered next).
- **Nested Modeltime Workflow:** This stage is where we create many models, fit the models to the data, and generate forecasts at scale.



Step 2: Preparing Nested Time Series Data Frame

There are three major steps in reparing the nested time series data frame. They are:

- Creating an initial data frame and extending to the future,
- Transforming the tibble data frame into nested modeltime data frame, and
- Splitting the nested data frame into training and test (hold-out) data sets.

Creating initial data frame and extending to the future

Firstly, we will create a new data table and extend the time frame 60 days into the future by using `extend_timeseries()` of modeltime.

```
nested_tbl <- grocery_cluster %>%
  extend_timeseries(
    .id_var = id,
    .date_var = date,
    .length_future = "60 days")
```

id	date	value
1	2015-01-01	3125.000
2	2015-01-01	0.000
3	2015-01-01	0.000
4	2015-01-01	0.000
5	2015-01-01	0.000
6	2015-01-01	0.000
7	2015-01-01	0.000
8	2015-01-01	0.000
9	2015-01-01	0.000
10	2015-01-01	0.000
11	2015-01-01	0.000
12	2015-01-01	0.000
13	2015-01-01	0.000
14	2015-01-01	0.000
15	2015-01-01	0.000
16	2015-01-01	0.000
17	2015-01-01	0.000

Nesting the tibble data frame

Next, `nest_timeseries()` is used to transform the newly created data frame in previous slide into a nested data frame by grouping the values in the *id* field.

```
nested_tbl <- nested_tbl %>%
    nest_timeseries(
      .id_var        = id,
      .length_future = 60,
      .length_actual = 17272)
```

Notice that the nested data frame consists of three fields namely *id*, *.actual_data* and *.future_data*.

id	.actual_data	.future_data
1	*1 variable*	*1 variable*
2	*1 variable*	*1 variable*
3	*1 variable*	*1 variable*
4	*1 variable*	*1 variable*
5	*1 variable*	*1 variable*
6	*1 variable*	*1 variable*
7	*1 variable*	*1 variable*
8	*1 variable*	*1 variable*
9	*1 variable*	*1 variable*
10	*1 variable*	*1 variable*
11	*1 variable*	*1 variable*
12	*1 variable*	*1 variable*
13	*1 variable*	*1 variable*
14	*1 variable*	*1 variable*
15	*1 variable*	*1 variable*
16	*1 variable*	*1 variable*
17	*1 variable*	*1 variable*

Data sampling

Lastly, `split_nested_timeseries()` is used to split the original data into training and testing (or hold-out) data sets.

```
nested_tbl <- nested_tbl %>%
  split_nested_timeseries(
    .length_test = 60)
```

id	.actual_data	.future_data	.splits
1	1 variable	1 variable	list(idx_train = 1:896, idx_test = 897:956)
2	1 variable	1 variable	list(idx_train = 1:896, idx_test = 897:956)
3	1 variable	1 variable	list(idx_train = 1:896, idx_test = 897:956)
4	1 variable	1 variable	list(idx_train = 1:896, idx_test = 897:956)
5	1 variable	1 variable	list(idx_train = 1:896, idx_test = 897:956)
6	1 variable	1 variable	list(idx_train = 1:896, idx_test = 897:956)
7	1 variable	1 variable	list(idx_train = 1:896, idx_test = 897:956)
8	1 variable	1 variable	list(idx_train = 1:896, idx_test = 897:956)
9	1 variable	1 variable	list(idx_train = 1:896, idx_test = 897:956)
10	1 variable	1 variable	list(idx_train = 1:896, idx_test = 897:956)
11	1 variable	1 variable	list(idx_train = 1:896, idx_test = 897:956)
12	1 variable	1 variable	list(idx_train = 1:896, idx_test = 897:956)
13	1 variable	1 variable	list(idx_train = 1:896, idx_test = 897:956)
14	1 variable	1 variable	list(idx_train = 1:896, idx_test = 897:956)
15	1 variable	1 variable	list(idx_train = 1:896, idx_test = 897:956)
16	1 variable	1 variable	list(idx_train = 1:896, idx_test = 897:956)
17	1 variable	1 variable	list(idx_train = 1:896, idx_test = 897:956)

Step 3: Creating Tidymodels Workflows

In this step, we will first applying tidymodels approach to create four forecasting models by using `recipe()` of **recipe** package and `workflow()` of **workflow** package. Both packages are member of **tidymodels**, a family of R packages specially designed for modeling and machine learning using **tidyverse** principles.

Model 1: Exponential Smoothing (Modeltime)

An Error-Trend-Season (ETS) model by using `exp_smoothing()`.

```
rec_autoETS <- recipe(
  value ~ date,
  extract_nested_train_split(
    nested_tbl))

wflw_autoETS <- workflow() %>%
  add_model(
    exp_smoothing() %>%
      set_engine("ets")) %>%
  add_recipe(rec_autoETS)
```

Model 2: Auto ARIMA (Modeltime)

An auto ARIMA model by using `arima_reg()`.

```
rec_autoARIMA <- recipe(
  value ~ date,
  extract_nested_train_split(
    nested_tbl))

wflw_autoARIMA <- workflow() %>%
  add_model(
    arima_reg() %>%
      set_engine("auto_arima")) %>%
  add_recipe(rec_autoARIMA)
```

Step 3: Creating Tidymodels Workflows (cont')

Model 3: Boosted Auto ARIMA (Modeltime)

An Boosted auto ARIMA model by using `arima_boost()`.

```
rec_xgb <- recipe(
  value ~ .,
  extract_nested_train_split(
    nested_tbl)) %>%
  step_timeseries_signature(date) %>%
  step_rm(date) %>%
  step_zv(all_predictors()) %>%
  step_dummy(all_nominal_predictors(),
             one_hot = TRUE)

wflw_xgb <- workflow() %>%
  add_model(
    boost_tree(
      "regression") %>%
      set_engine("xgboost")) %>%
  add_recipe(rec_xgb)
```

Model 4: prophet (Modeltime)

A prophet model using `prophet_reg()`.

```
rec_prophet <- recipe(
  value ~ date,
  extract_nested_train_split(
    nested_tbl))

wflw_prophet <- workflow() %>%
  add_model(
    prophet_reg(
      "regression",
      seasonality_yearly = TRUE) %>%
      set_engine("prophet")
    ) %>%
  add_recipe(rec_prophet)
```

Nested Forecasting with modeltime

Core Functions | Nested Forecasting

1: Nested Fitting

`modeltime_nested_fit()`

- **Trains** each model on training split
- **Logs** test accuracy, test forecast with confidence intervals on testing split
- **Logs** additional information including error reports

2: Select Best

`modeltime_nested_select_best()`

- **Selects best model** using accuracy metric
- **Filters** test forecasts to just those of best models
- **Logs** best models

3: Nested Refitting

`modeltime_nested_refit()`

- **Retrains** selected models on actual data
- **Logs** future forecast on future data

Step 4: Fitting Nested Forecasting Models

In this step, `modeltime_nested_fit()` is used to fit the four models we created in Step 3. Note that the input must be in the form of nested modeltime table (i.e. *nested_tbl*)

```
nested_tbl <- modeltime_nested_fit(
  nested_data = nested_tbl,
  wflw_autoETS,
  wflw_autoARIMA,
  wflw_prophet,
  wflw_xgb)
```

The nested modeltime data frame

The output object *nested_tbl* is a nested modetime data frame.

id	.actual_data	.future_data	.splits	.modeltime_tables
1	*1 variable*	*1 variable*	*list(idx_train = 1:896, idx_test = 897:956)*	*3 variables*
2	*1 variable*	*1 variable*	*list(idx_train = 1:896, idx_test = 897:956)*	*3 variables*
3	*1 variable*	*1 variable*	*list(idx_train = 1:896, idx_test = 897:956)*	*3 variables*
4	*1 variable*	*1 variable*	*list(idx_train = 1:896, idx_test = 897:956)*	*3 variables*
5	*1 variable*	*1 variable*	*list(idx_train = 1:896, idx_test = 897:956)*	*3 variables*

Click on the table icon of the first row under *.modeltime_tables* field, its corresponding modelling data frame appears.

.model_id	.model	.model_desc	.type	.calibration_data
1	*list(pre = list(actions = list(recipe = list(recip [...]*	ETSMNA	Test	*1 variable*
2	*list(pre = list(actions = list(recipe = list(recip [...]*	ARIMA	Test	*1 variable*
3	*list(pre = list(actions = list(recipe = list(recip [...]*	PROPHET	Test	*1 variable*
4	*list(pre = list(actions = list(recipe = list(recip [...]*	XGBOOST	Test	*1 variable*

Notice that the four models were fitted by using the test data set.

Step 5: Model Accuracy Assessment with Test Logged Attributes

Before we go ahead to select the best model, it is a good practice to compare the performance of the models by using accuracy matrices.

```
nested_tbl %>%
  extract_nested_test_accuracy() %>%
  table_modeltime_accuracy(
    .interactive = FALSE)
```

What can we learn fro mthe code chunk above?

- `extract_nested_test_accuracy()` is used to extract the accuracy matrices by using the test data set.
- `table_modeltime_accuracy()` is used to display the accuracy report in tabular form.

Note that `.interactive` argument returns interactive or static tables. If TRUE, returns `reactable` table. If FALSE, returns static `gt` table.

Step 5: Model Accuracy Assessment with Test Logged Attributes

Accuracy Table

id	.model_id	.model_desc	.type	mae	mape	mase	smape	rmse	rsq
1	1	ETSMNA	Test	980.30	9.44	0.59	9.66	1275.14	0.52
1	2	ARIMA	Test	1297.01	13.00	0.78	12.87	1602.60	0.11
1	3	PROPHET	Test	869.20	8.43	0.52	8.53	1061.91	0.60
1	4	XGBOOST	Test	774.08	7.47	0.46	7.53	964.33	0.66
2	1	ETSANA	Test	749.22	11.58	0.82	10.48	1116.95	0.21
2	2	ARIMA	Test	1103.26	17.13	1.21	15.41	1342.42	0.00
2	3	PROPHET	Test	633.69	9.24	0.69	9.04	982.70	0.29
2	4	XGBOOST	Test	497.78	7.82	0.54	7.28	678.29	0.73
3	1	ETSANA	Test	2349.65	12.20	0.85	12.03	2929.23	0.33
3	2	ARIMA	Test	3056.48	14.87	1.11	15.60	3817.20	0.01
3	3	PROPHET	Test	2211.61	11.59	0.80	11.26	2593.66	0.49
3	4	XGBOOST	Test	2113.59	10.77	0.77	10.88	2622.32	0.52
4	1	ETSANA	Test	1022.66	9.59	0.50	9.30	1380.97	0.66

Step 6: Extracting and Visualising Nested Test Forecast

In this step, `extract_nested_test_forecast()` is used to extract the forecasted values from the nested modeltime data frame and `plot_modeltime_forecast()` is used to plot the forecasted values graphically.

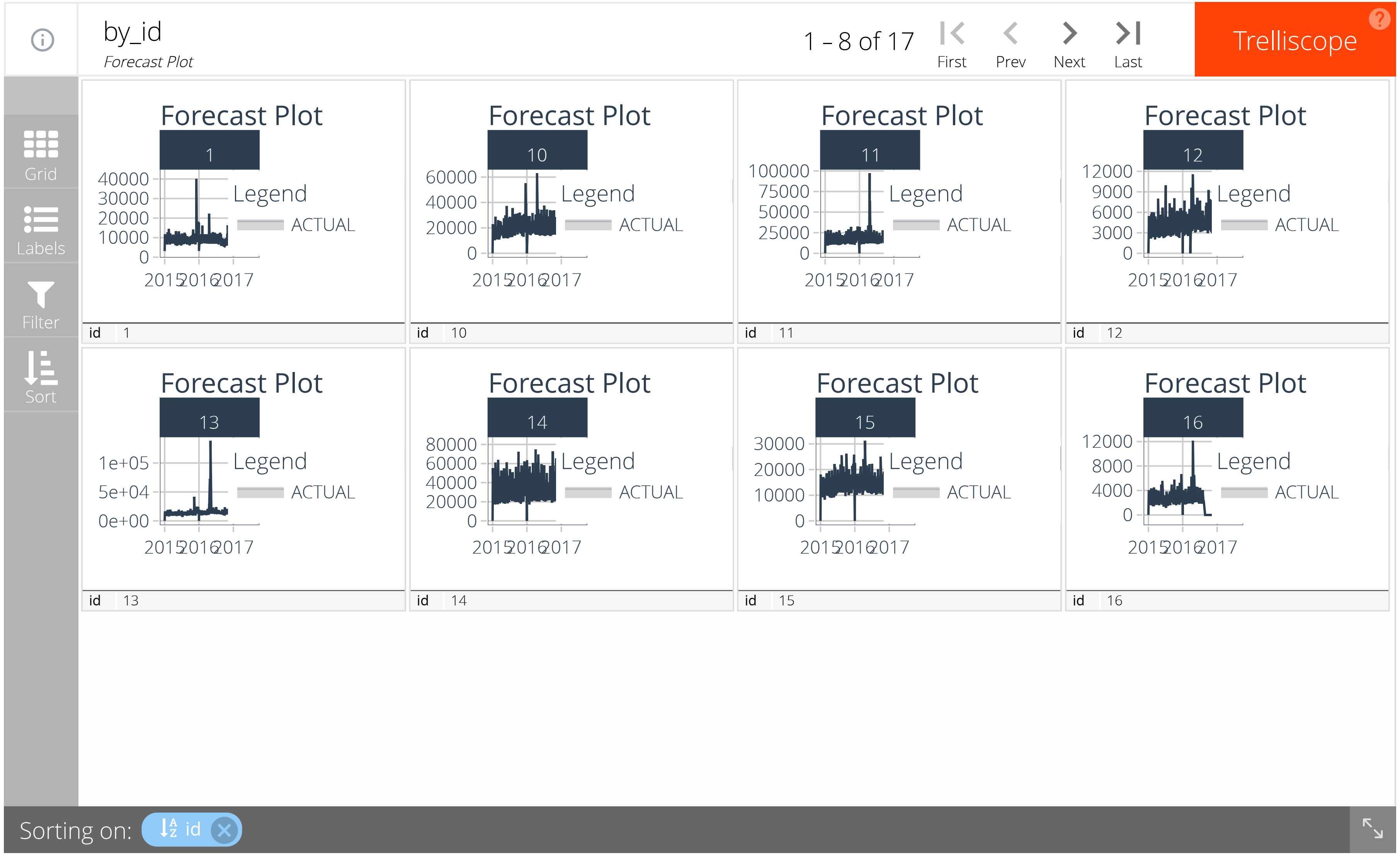
```
nested_tbl %>%
  extract_nested_test_forecast() %>%
  group_by(id) %>%
  plot_modeltime_forecast(
    .facet_ncol = 4,
    .interactive = FALSE)
```

Static multiple small line graphs

Interactive multiple small line graphs with tralliscopejs

```
nested_tbl %>%
  extract_nested_test_forecast() %>%
  group_by(id) %>%
  plot_modeltime_forecast(
    .line_size = 0.4,
    .facet_ncol = 4,
    .facet_nrow = 2,
    .facet_scales = "free_y",
    .interactive = TRUE,
    .smooth_size = 0.4,
    .trelliscope = TRUE,
    .trelliscope_params = list(
      width = 640,
      height = 480,
      path= "trellis4/")
    )
```

Interactive multiple small line graphs with tralliscopejs



Step 7: Extracting nested error logs

Before going ahead to choose the best model, it is always a good practice to examine if there is any error in the model. This task can be accomplished by using `extract_nested_error_report()`.

```
nested_tbl %>%
  extract_nested_error_report()
```

```
## # A tibble: 0 × 4
## # … with 4 variables: id <fct>, .model_id <int>, .model_desc <chr>,
## #   .error_desc <chr>
```

Step 8: Selecting the Best Model

Now we are ready to select the best model by using `modeltime_nested_select_best()`.

```
best_nested_tbl <- nested_tbl %>%
  modeltime_nested_select_best(
    metric = "rmse",
    minimize = TRUE,
    filter_test_forecasts = TRUE)
```

Note that to select the best forecasting models, the `minimize` argument must set to *TRUE*.

Extracting and displaying nested best model report

After selecting the best model for each time series, we can display the best model report by using `extract_nested_best_model_report()`.

```
best_nested_tbl %>%
  extract_nested_best_model_report() %>%
  table_modeltime_accuracy(
    .interactive = FALSE)
```

Extracting and displaying nested best model report

Accuracy Table

id	.model_id	.model_desc	.type	mae	mape	mase	smape	rmse	rsq
1	4	XGBOOST	Test	774.08	7.47	0.46	7.53	964.33	0.66
2	4	XGBOOST	Test	497.78	7.82	0.54	7.28	678.29	0.73
3	3	PROPHET	Test	2211.61	11.59	0.80	11.26	2593.66	0.49
4	4	XGBOOST	Test	760.45	7.32	0.37	7.11	988.27	0.82
5	4	XGBOOST	Test	799.55	7.31	0.36	7.14	1146.20	0.83
6	4	XGBOOST	Test	2867.44	8.46	0.58	8.29	3647.97	0.81
7	4	XGBOOST	Test	539.32	9.72	0.71	9.82	646.86	0.52
8	4	XGBOOST	Test	1241.69	7.32	0.55	6.93	1649.55	0.73
9	4	XGBOOST	Test	412.86	7.35	0.48	7.26	507.81	0.75
10	4	XGBOOST	Test	2126.00	8.47	0.41	8.68	2658.25	0.83
11	4	XGBOOST	Test	2013.79	8.01	0.41	7.83	2535.41	0.78
12	3	PROPHET	Test	641.66	14.40	0.82	13.29	747.16	0.65
13	4	XGBOOST	Test	850.80	5.79	0.46	5.59	1110.86	0.81

Extracting and Visualising Nested Best Test Forecasts

We can also plot multiple small line graphs by using `plot_modeltime_forecast()`.

```
best_nested_tbl %>%
  extract_nested_test_forecast() %>%
  group_by(id) %>%
  plot_modeltime_forecast(
    .facet_ncol = 4,
    .interactive = FALSE)
```

Extracting and Visualising Nested Best Test Forecasts

Step 9: Refitting and forecast forward

The last step of the forecasting process is to refit the best models with the full data set and forecast to the future by using `modeltime_nested_refit()`.

```
nested_refit_tbl <- best_nested_tbl %>%
  modeltime_nested_refit(
    control = control_nested_refit(
      verbose = TRUE))
```

Note that `control_nested_refit(verbose = TRUE)` is used to display the modelling results as each model is refit. This is an useful way to follow the nested model fitting process.

Extracting and Visualising Nested Future Forecast

Similar, `plot_modeltime_forecast()` can be used to visualise the forecasts. However, instead of `extracted_nested_test_forecast()` is used, `extract_nested_future_forecast()` is used.

```
nested_refit_tbl %>%
  extract_nested_future_forecast() %>%
  group_by(id) %>%
  plot_modeltime_forecast(
    .interactive = FALSE,
    .facet_ncol = 4)
```

Extracting and Visualising Nested Future Forecast

Interactive Line Graph of Future Forecast

```r
nested_refit_tbl %>%
  extract_nested_future_forecast() %>%
  filter(id == 1) %>%
  plot_modeltime_forecast(
    .interactive = TRUE,
    .facet_ncol = 4,
    .plotly_slider = TRUE)
```