

Package ‘forecastML’

January 7, 2020

Type Package

Title Time Series Forecasting with Machine Learning Methods

Version 0.7.0

Author Nickalus Redell

Maintainer Nickalus Redell <nickalusredell@gmail.com>

Description The purpose of 'forecastML' is to simplify the process of multi-step-ahead direct forecasting with standard machine learning algorithms. 'forecastML' supports lagged, dynamic, static, and grouping features for modeling single and grouped numeric or factor/sequence time series. In addition, simple wrapper functions are used to support model-building with most R packages. This approach to forecasting is inspired by Bergmeir, Hyndman, and Koo's (2018) paper "A note on the validity of cross-validation for evaluating autoregressive time series prediction" <doi:10.1016/j.csda.2017.11.003>.

License MIT + file LICENSE

URL <https://github.com/nredell/forecastML/>

Encoding UTF-8

LazyData true

Imports tidyr (>= 0.8.1), rlang (>= 0.4.0), magrittr (>= 1.5), lubridate (>= 1.7.4), ggplot2 (>= 3.1.0), future.apply (>= 1.3.0), methods, purrr (>= 0.3.2), data.table (>= 1.12.6), dplyr (>= 1.0.0)

RoxygenNote 6.1.1

Collate 'fill_gaps.R' 'create_windows.R' 'combine_forecasts.R' 'lagged_df.R' 'return_error.R' 'return_hyper.R' 'train_model.R' 'data_seatbelts.R' 'data_buoy.R' 'data_buoy_gaps.R' 'zzz.R'

Depends R (>= 3.4.0), dplyr (>= 0.8.3)

Suggests glmnet (>= 2.0.16), DT (>= 0.5), knitr (>= 1.22), rmarkdown (>= 1.12.6), xgboost (>= 0.82.1), randomForest (>= 4.6.14), testthat (>= 2.2.1), covr (>= 3.3.1)

VignetteBuilder knitr

NeedsCompilation no

Repository CRAN

Date/Publication 2020-01-07 10:30:03 UTC

R topics documented:

combine_forecasts	2
create_lagged_df	4
create_windows	8
data_buoy	9
data_buoy_gaps	10
data_seatbelts	11
fill_gaps	11
plot.forecastML	13
plot.forecast_model_hyper	14
plot.forecast_results	16
plot.lagged_df	17
plot.training_results	18
plot.validation_error	18
plot.windows	19
predict.forecast_model	20
return_error	23
return_hyper	25
summary.lagged_df	27
train_model	28

Index	31
--------------	-----------

combine_forecasts	<i>Combine multiple horizon-specific forecast models to produce one forecast</i>
-------------------	--

Description

The horizon-specific models can either be combined to (a) produce final forecasts for only those horizons at which they were trained (i.e., shorter-horizon models override longer-horizon models when producing final short-horizon h-step-ahead forecasts) or (b) produce final forecasts using any combination of horizon-specific models that minimized error over the validation/training dataset.

Usage

```
combine_forecasts(..., type = c("horizon", "error"),
  data_error = list(NULL), metric = NULL)
```

Arguments

... One or more objects of class 'forecast_results' from running `predict.forecast_model()` on an input forward-looking forecast dataset. These are the forecasts from the horizon-specific direct forecasting models trained over the entire training dataset by setting `create_windows(..., window_length = 0)`. If multiple models are passed in ..., the model names from `train_model()` should be unique for a given model forecast horizon.

type	Default: 'horizon'. A character vector of length 1 that identifies the forecast combination method.
data_error	Optional. A list of objects of class 'validation_error' from running <code>return_error()</code> on a training dataset. The length and order of <code>data_error</code> should match the models passed in
metric	Required if <code>data_error</code> is given. A length 1 character vector naming the forecast error metric used to select the optimal model at each forecast horizon from the models passed in '...' e.g., 'mae'.

Value

An S3 object of class 'forecastML' with final h-step-ahead forecasts.

Forecast combination type:

- `type = 'horizon'`: 1 final h-step-ahead forecast is returned for each model object passed in
- `type = 'error'`: 1 final h-step-ahead forecast is returned by selecting, for each forecast horizon, the model that minimized the chosen error metric at that horizon on the outer-loop validation data sets.

Columns in returned 'forecastML' data.frame:

- `model`: User-supplied model name in `train_model()`.
- `model_forecast_horizon`: The direct-forecasting time horizon that the model was trained on.
- `horizon`: Forecast horizons, 1:h, measured in dataset rows.
- `forecast_period`: The forecast period in row indices or dates. The forecast period starts at either `attributes(create_lagged_df())$data_stop + 1` for row indices or `attributes(create_lagged_df())$d + 1 * frequency` for date indices.
- `"groups"`: If given, the user-supplied groups in `create_lagged_df()`.
- `"outcome_name"_pred`: The final forecasts.
- `"outcome_name"_pred_lower`: If given, the lower forecast bounds returned by the user-supplied prediction function.
- `"outcome_name"_pred_upper`: If given, the upper forecast bounds returned by the user-supplied prediction function.

Methods and related functions

The output of `combine_forecasts()` has the following generic S3 methods

- `plot`

Examples

```

# Example with "type = 'horizon'".
data("data_seatbelts", package = "forecastML")

horizons <- c(1, 3, 12)
lookback <- 1:15

data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                              lookback = lookback, horizon = horizons)

windows <- create_windows(data_train, window_length = 0)

model_function <- function(data, my_outcome_col) {
  model <- lm(DriversKilled ~ ., data = data)
  return(model)
}

model_results <- train_model(data_train, windows, model_name = "LM", model_function)

data_forecast <- create_lagged_df(data_seatbelts, type = "forecast", outcome_col = 1,
                                 lookback = lookback, horizon = horizons)

prediction_function <- function(model, data_features) {
  x <- data_features
  data_pred <- data.frame("y_pred" = predict(model, newdata = x))
  return(data_pred)
}

data_forecasts <- predict(model_results, prediction_function = list(prediction_function),
                          data = data_forecast)

data_combined <- combine_forecasts(data_forecasts)

plot(data_combined)

```

create_lagged_df	<i>Create model training and forecasting datasets with lagged, grouped, dynamic, and static features</i>
------------------	--

Description

Create a list of datasets with lagged, grouped, dynamic, and static features to (a) train forecasting models for specified forecast horizons and (b) forecast into the future with a trained ML model.

Usage

```

create_lagged_df(data, type = c("train", "forecast"), outcome_col = 1L,
                 horizons, lookback = NULL, lookback_control = NULL, dates = NULL,
                 frequency = NULL, dynamic_features = NULL, groups = NULL,
                 static_features = NULL, use_future = FALSE, keep_rows = FALSE)

```

Arguments

data	A data.frame with the (a) target to be forecasted and (b) features/predictors. An optional date column can be given in the dates argument (required for grouped time series). Note that forecastML only works with regularly spaced date/time intervals and that missing rows—usually due to periods when no data was collected—will result in poorly trained models due to incorrect feature lags. Use fill_gaps to fill in any missing rows/data prior to running this function.
type	The type of dataset to return—(a) model training or (b) forecast prediction. The default is train.
outcome_col	The column index—an integer—of the target to be forecasted. Forecasting only one outcome column is allowed at present, however, groups of time series can be forecasted if they are stacked vertically in a long dataset and the groups, dates, and frequency arguments are specified.
horizons	A numeric vector of one or more forecast horizons, h, measured in input dataset rows. For each horizon, 1:h forecasts are returned (e.g., horizons = 12 trains a model to minimize 1 to 12-step-ahead error and returns forecasts for 1:12 steps into the future). If dates are given, a horizon of 1, for example, would equal 1 * frequency in calendar time.
lookback	A numeric vector giving the lags—in dataset rows—for creating the lagged features. All non-grouping, non-static, and non-dynamic features in the input dataset, data, are lagged by the same values. For lagged features, lag values that don't support direct forecasting for a given forecast horizon (e.g, a lookback of 3 for a 4-step-ahead horizon) are silently dropped from the data.frame. Either lookback or lookback_control need to be specified.
lookback_control	A list of numeric vectors, specifying potentially unique lags for each feature. The length of the list should equal ncol(data) and be ordered the same as the columns in data. Lag values for any grouping, static, or dynamic feature columns are automatically coerced to 0 and not lagged. list(NULL) lookback_control values drop columns from the input dataset. For lagged features, lag values that don't support direct forecasting for a given forecast horizon are silently dropped from the data.frame. Either lookback or lookback_control need to be specified.
dates	A vector or 1-column data.frame of dates/times with class 'Date' or 'POSIXt'. The length of 'dates' should equal nrow(data). Required if groups are given.
frequency	Date/time frequency. Required if dates are given. A string taking the same input as base::seq.Date(...,by="frequency") or base::seq.POSIXt(...,by="frequency") e.g., '1 hour', '1 month', '7 days', '10 years' etc. The highest frequency supported at present is '1 sec'.
dynamic_features	A character vector of column names that identify features that change through time but which are not lagged (e.g., weekday or year). If type = "forecast", these features will receive NA values; though, they can be filled in by the user after running this function.
groups	A character vector of column names that identify the groups/hierarchies when multiple time series are present. These columns are used as model features but

are not lagged. Note that combining feature lags with grouped time series will result in NA values throughout the data.

static_features	For grouped time series only. A character vector of column names that identify features that do not change through time. These columns are not lagged. If type = "forecast", these features will be filled forward using the most recent value for the group.
use_future	Boolean. If TRUE, the future package is used for creating lagged data.frames. multisession or multicore futures are especially useful for (a) grouped time series with many groups and (b) high-dimensional datasets with many lags per feature. Run future::plan(future::multiprocess) prior to this function to set up multisession or multicore parallel dataset creation.
keep_rows	Boolean. For non-grouped time series, keep the 1:max(lookback) rows at the beginning of the time series. These rows will contain missing values for lagged features that "look back" before the start of the dataset.

Value

An S3 object of class 'lagged_df' or 'grouped_lagged_df': A list of data.frames with new columns for the lagged/non-lagged features. The length of the returned list is equal to the number of forecast horizons and is in the order of horizons supplied to the horizons argument. Horizon-specific datasets can be accessed with my_lagged_df\$horizon_h where 'h' gives the forecast horizon.

The contents of the returned data.frames are as follows:

type = 'train', non-grouped: A data.frame with the outcome and lagged/dynamic features.

type = 'train', grouped: A data.frame with the outcome and unlagged grouping columns followed by lagged, dynamic, and static features.

type = 'forecast', non-grouped: (1) An 'index' column giving the row index or date of the forecast periods (e.g., a 100 row non-date-based training dataset would start with an index of 101). (2) A 'horizon' column that indicates the forecast period from 1:max(horizons). (3) Lagged features identical to the 'train', non-grouped dataset.

type = 'forecast', grouped: (1) An 'index' column giving the date of the forecast periods. The first forecast date for each group is the maximum date from the dates argument + 1 * frequency which is the user-supplied date/time frequency. (2) A 'horizon' column that indicates the forecast period from 1:max(horizons). (3) Lagged, static, and dynamic features identical to the 'train', grouped dataset.

Attributes

- names: The horizons-specific datasets that can be accessed by my_lagged_df\$horizon_h where 'h' gives the forecast horizons.
- type: Training, train, or forecasting, forecast, dataset(s).
- horizons: Forecast horizons measured in dataset rows.
- outcome_col: The column index of the target being forecasted.
- outcome_names: The name of the target being forecasted.
- predictor_names: The predictor or feature names from the input dataset.

- `row_indices`: The `row.names()` of the output dataset. For non-grouped datasets, the first `lookback + 1` rows are removed from the beginning of the dataset to remove NA values in the lagged features.
- `date_indices`: If dates are given, the vector of dates.
- `frequency`: If dates are given, the date/time frequency.
- `data_start`: `min(row_indices)` or `min(date_indices)`.
- `data_stop`: `max(row_indices)` or `max(date_indices)`.
- `groups`: If groups are given, a vector of group names.
- `class`: `grouped_lagged_df`, `lagged_df`, `list`

Methods and related functions

The output of `create_lagged_df()` is passed into

- [create_windows](#)

and has the following generic S3 methods

- [summary](#)
- [plot](#)

Examples

```
# Sampled Seatbelts data from the R package datasets.
data("data_seatbelts", package = "forecastML")
#-----
# Example 1 - Training data for 2 horizon-specific models w/ common lags per predictor.
horizons <- c(1, 12)
lookback <- 1:15

data <- data_seatbelts

data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                              horizons = horizons, lookback = lookback)
head(data_train[[length(horizons)]])

# Example 1 - Forecasting dataset
# The last 'nrow(data_seatbelts) - horizon' rows are automatically used from data_seatbelts.
data_forecast <- create_lagged_df(data_seatbelts, type = "forecast", outcome_col = 1,
                                  horizons = horizons, lookback = lookback)
head(data_forecast[[length(horizons)]])

#-----
# Example 2 - Training data for one 3-month horizon model w/ unique lags per predictor.
horizons <- 3
lookback <- list(c(3, 6, 9, 12), c(4:12), c(6:15), c(8))

data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                              horizons = horizons, lookback_control = lookback)
head(data_train[[length(horizons)]])
```

 create_windows

Create time-contiguous validation datasets for model evaluation

Description

Flexibly ceate blocks of time-contiguous validation datasets to assess forecast accuracy of trained models at various times in the past. These validation datasets are similar to the outer loop of a nested cross-validation model training setup.

Usage

```
create_windows(lagged_df, window_length = 12L, window_start = NULL,
              window_stop = NULL, skip = 0, include_partial_window = TRUE)
```

Arguments

lagged_df	An object of class 'lagged_df' or 'grouped_lagged_df' from create_lagged_df .
window_length	An integer that defines the length of the contiguous validation dataset in dataset rows/dates. If dates were given in <code>create_lagged_df()</code> , the validation window is 'window_length' * 'date frequency' in calendar time. Setting <code>window_length = 0</code> trains the model on (a) the entire dataset or (b) between a single <code>window_start</code> and <code>window_stop</code> value. Specifying multiple <code>window_start</code> and <code>window_stop</code> values with vectors of length > 1 overrides <code>window_length</code> .
window_start	Optional. A row index or date identifying the row/date to start creating contiguous validation datasets. A vector of start rows/dates can be supplied for greater control. The length and order of <code>window_start</code> should match <code>window_stop</code> . If <code>length(window_start) > 1</code> , <code>window_length</code> , <code>skip</code> , and <code>include_partial_window</code> are ignored.
window_stop	Optional. An index or date identifying the row/date to stop creating contiguous validation datasets. A vector of start rows/dates can be supplied for greater control. The length and order of <code>window_stop</code> should match <code>window_start</code> . If <code>length(window_stop) > 1</code> , <code>window_length</code> , <code>skip</code> , and <code>include_partial_window</code> are ignored.
skip	An integer giving a fixed number of dataset rows/dates to skip between validation datasets. If dates were given in <code>create_lagged_df()</code> , the time between validation windows is <code>skip * 'date frequency'</code> .
include_partial_window	Boolean. If TRUE, keep validation datasets that are shorter than <code>window_length</code> .

Value

An S3 object of class 'windows': A data.frame giving the indices for the validation datasets.

Methods and related functions

The output of `create_windows()` is passed into

- `train_model`

and has the following generic S3 methods

- `plot`

Examples

```
# Sampled Seatbelts data from the R package datasets.
data("data_seatbelts", package = "forecastML")

# Example - Training data for 2 horizon-specific models w/ common lags per feature.
horizons <- c(1, 12)
lookback <- 1:15

data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                              lookback = lookback, horizon = horizons)

# All historical window lengths of 12 plus any partial windows at the end of the dataset.
windows <- create_windows(data_train, window_length = 12)
windows

# Two custom validation windows with different lengths.
windows <- create_windows(data_train, window_start = c(20, 80), window_stop = c(30, 100))
windows
```

data_buoy

NOAA buoy weather data

Description

A dataset containing daily average sensor measurements of several environmental conditions collected by 14 buoys in Lake Michigan from 2012 through 2018.

Usage

```
data_buoy
```

Format

A data.frame with 30,821 rows and 9 columns:

date date

wind_spd average daily wind speed in kts

buoy_id the station ID for each buoy

lat latitude
lon longitude
day day of year
year calendar year
air_temperature air temperature in degrees Fahrenheit
sea_surface_temperature water temperature in degrees Fahrenheit

Source

<http://www.ndbc.noaa.gov/>

data_buoy_gaps	<i>NOAA buoy weather data</i>
----------------	-------------------------------

Description

A dataset containing daily average sensor measurements of several environmental conditions collected by 14 buoys in Lake Michigan from 2012 through 2018. This dataset is identical to the [data_buoy](#) dataset except that there are gaps in the daily sensor data. Running `fill_gaps()` on `data_buoy_gaps` will produce `data_buoy`.

Usage

`data_buoy_gaps`

Format

A data.frame with 23,646 rows and 9 columns:

date date
wind_spd average daily wind speed in kts
buoy_id the station ID for each buoy
lat latitude
lon longitude
day day of year
year calendar year
air_temperature air temperature in degrees Fahrenheit
sea_surface_temperature water temperature in degrees Fahrenheit

Source

<http://www.ndbc.noaa.gov/>

data_seatbelts	<i>Road Casualties in Great Britain 1969-84</i>
----------------	---

Description

This is the Seatbelts dataset from the datasets package.

Usage

```
data_seatbelts
```

Format

A data.frame with 192 rows and 8 columns

Source

Harvey, A.C. (1989). Forecasting, Structural Time Series Models and the Kalman Filter. Cambridge University Press, pp. 519–523.

Durbin, J. and Koopman, S. J. (2001). Time Series Analysis by State Space Methods. Oxford University Press.

<https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/UKDriverDeaths.html>

fill_gaps	<i>Prepare a dataset for modeling by filling in temporal gaps in data collection</i>
-----------	--

Description

In order to create a modeling dataset with feature lags that are temporally correct, the entry function in forecastML, `create_lagged_df`, needs evenly-spaced time series with no gaps in data collection. `fill_gaps()` can help here. This function takes a data.frame with (a) dates, (b) the outcome being forecasted, and, optionally, (c) dynamic features that change through time, (d) group columns for multiple time series modeling, and (e) static or non-dynamic features for multiple time series modeling and returns a data.frame with rows evenly spaced in time. Specifically, this function adds rows to the input dataset while filling in (a) dates, (b) grouping information, and (c) static features. The (a) outcome and (b) dynamic features will be NA for any missing time periods; these NA values can be left as-is, user-imputed, or removed from modeling in the user-supplied modeling wrapper function for `train_model`.

Usage

```
fill_gaps(data, date_col = 1L, frequency, groups = NULL,  
          static_features = NULL)
```

Arguments

<code>data</code>	A <code>data.frame</code> or object coercible to a <code>data.frame</code> with, minimally, dates and the outcome being forecasted.
<code>date_col</code>	The column index—an integer—of the date index. This column should have class 'Date' or 'POSIXt'.
<code>frequency</code>	Date/time frequency. A string taking the same input as <code>base::seq.Date(..., by = "frequency")</code> or <code>base::seq.POSIXt(..., by = "frequency")</code> e.g., '1 hour', '1 month', '7 days', '10 years' etc. The highest frequency supported at present is '1 sec'.
<code>groups</code>	Optional. A character vector of column names that identify the unique time series (i.e., groups/hierarchies) when multiple time series are present.
<code>static_features</code>	Optional. For grouped time series only. A character vector of column names that identify features that do not change through time. These columns are expected to be used as model features but are not lagged (e.g., a ZIP code column). The most recent values for each static feature for each group are used to fill in the resulting missing data in static features when new rows are added to the dataset.

Value

An object of class 'data.frame': The returned `data.frame` has the same number of columns and column order but with additional rows to account for gaps in data collection. For grouped data, any new rows added to the returned `data.frame` will appear between the minimum—or oldest—date for that group and the maximum—or most recent—date across all groups. If the user-supplied forecasting algorithm(s) cannot handle missing outcome values or missing dynamic features, these should either be imputed prior to `create_lagged_df()` or filtered out in the user-supplied modeling function for [train_model](#).

Methods and related functions

The output of `fill_gaps()` is passed into

- [create_lagged_df](#)

Examples

```
# NOAA buoy dataset with gaps in data collection
data("data_buoy_gaps", package = "forecastML")

data_buoy_no_gaps <- fill_gaps(data_buoy_gaps, date_col = 1, frequency = '1 day',
                              groups = 'buoy_id', static_features = c('lat', 'lon'))

# The returned data.frame has the same number of columns but the time-series
# are now evenly spaced at 1 day apart. Additionally, the unchanging grouping
# columns and static features columns have been filled in for the newly created dataset rows.
dim(data_buoy_gaps)
dim(data_buoy_no_gaps)

# Running create_lagged_df() is the next step in the forecastML forecasting
```

```
# process. If there are long gaps in data collection, like in this buoy dataset,
# and the user-supplied modeling algorithm cannot handle missing outcomes data,
# the best option is to filter these rows out in the user-supplied modeling function
# for train_model()
```

plot.forecastML *Plot an object of class 'forecastML'*

Description

A forecast plot of h-step-ahead forecasts produced from multiple horizon-specific forecast models using `combine_forecasts()`.

Usage

```
## S3 method for class 'forecastML'
plot(x, data_actual = NULL, actual_indices = NULL,
     models = NULL, group_filter = NULL, drop_facet = FALSE, ...)
```

Arguments

<code>x</code>	An object of class 'forecastML' from <code>combine_forecasts()</code> .
<code>data_actual</code>	A data.frame containing the target/outcome name and any grouping columns. The data can be historical actuals and/or holdout/test data.
<code>actual_indices</code>	Required if <code>data_actual</code> is given. A vector or 1-column data.frame of numeric row indices or dates (class 'Date' or 'POSIXt') with length <code>nrow(data_actual)</code> . The data can be historical actuals and/or holdout/test data.
<code>models</code>	Optional. Filter results by user-defined model name from <code>train_model()</code> .
<code>group_filter</code>	Optional. A string for filtering plot results for grouped time-series (e.g., "group_col_1 == 'A'"); passed to <code>dplyr::filter()</code> internally.
<code>drop_facet</code>	Optional. Boolean. If actuals are given when forecasting factors, the plot facet with 'actual' data can be dropped.
<code>...</code>	Not used.

Value

Forecast plot of class 'ggplot'.

```
plot.forecast_model_hyper
  Plot hyperparameters
```

Description

Plot hyperparameter stability and relationship with error metrics across validation datasets and horizons.

Usage

```
## S3 method for class 'forecast_model_hyper'
plot(x, data_results, data_error,
     type = c("stability", "error"), horizons = NULL, windows = NULL,
     ...)
```

Arguments

x	An object of class 'forecast_model_hyper' from return_hyper().
data_results	An object of class 'training_results' from predict.forecast_model().
data_error	An object of class 'validation_error' from return_error().
type	Select plot type; 'stability' is the default.
horizons	Optional. A numeric vector to filter results by horizon.
windows	Optional. A numeric vector to filter results by validation window number.
...	Not used.

Value

Hyperparameter plots of class 'ggplot'.

Examples

```
# Sampled Seatbelts data from the R package datasets.
data("data_seatbelts", package = "forecastML")

# Example - Training data for 2 horizon-specific models w/ common lags per predictor.
horizons <- c(1, 12)
lookback <- 1:15

data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                              lookback = lookback, horizon = horizons)

# One custom validation window at the end of the dataset.
windows <- create_windows(data_train, window_start = 181, window_stop = 192)

# User-define model - LASSO
# A user-defined wrapper function for model training that takes the following
```

```

# arguments: (1) a horizon-specific data.frame made with create_lagged_df(..., type = "train")
# (e.g., my_lagged_df$horizon_h) and, optionally, (2) any number of additional named arguments
# which are passed as '...' in train_model().
library(glmnet)
model_function <- function(data, my_outcome_col) {

  x <- data[, -(my_outcome_col), drop = FALSE]
  y <- data[, my_outcome_col, drop = FALSE]
  x <- as.matrix(x, ncol = ncol(x))
  y <- as.matrix(y, ncol = ncol(y))

  model <- glmnet::cv.glmnet(x, y, nfolds = 3)
  return(model)
}

# my_outcome_col = 1 is passed in ... but could have been defined in model_function().
model_results <- train_model(data_train, windows, model_name = "LASSO", model_function,
                             my_outcome_col = 1)

# User-defined prediction function - LASSO
# The predict() wrapper takes two positional arguments. First,
# the returned model from the user-defined modeling function (model_function() above).
# Second, a data.frame of predictors--identical to the datasets returned from
# create_lagged_df(..., type = "train"). The function can return a 1- or 3-column data.frame
# with either (a) point forecasts or (b) point forecasts plus lower and upper forecast
# bounds (column order and column names do not matter).
prediction_function <- function(model, data_features) {

  x <- as.matrix(data_features, ncol = ncol(data_features))

  data_pred <- data.frame("y_pred" = predict(model, x, s = "lambda.min"))
  return(data_pred)
}

# Predict on the validation datasets.
data_valid <- predict(model_results, prediction_function = list(prediction_function),
                      data = data_train)

# User-defined hyperparameter function - LASSO
# The hyperparameter function should take one positional argument--the returned model
# from the user-defined modeling function (model_function() above). It should
# return a 1-row data.frame of the optimal hyperparameters.
hyper_function <- function(model) {

  lambda_min <- model$lambda.min
  lambda_1se <- model$lambda.1se

  data_hyper <- data.frame("lambda_min" = lambda_min, "lambda_1se" = lambda_1se)
  return(data_hyper)
}

data_error <- return_error(data_valid)

```

```
data_hyper <- return_hyper(model_results, hyper_function)

plot(data_hyper, data_valid, data_error, type = "stability", horizons = c(1, 12))
```

`plot.forecast_results` *Plot an object of class forecast_results*

Description

A forecast plot for each horizon for each model in `predict.forecast_model()`.

Usage

```
## S3 method for class 'forecast_results'
plot(x, data_actual = NULL,
     actual_indices = NULL, models = NULL, horizons = NULL,
     windows = NULL, facet_plot = c("model", "model_forecast_horizon"),
     group_filter = NULL, ...)
```

Arguments

<code>x</code>	An object of class 'forecast_results' from <code>predict.forecast_model()</code> .
<code>data_actual</code>	A data.frame containing the target/outcome name and any grouping columns. The data can be historical actuals and/or holdout/test data.
<code>actual_indices</code>	Required if <code>data_actual</code> is given. A vector or 1-column data.frame of numeric row indices or dates (class 'Date' or 'POSIXt') with length <code>nrow(data_actual)</code> . The data can be historical actuals and/or holdout/test data.
<code>models</code>	Optional. Filter results by user-defined model name from <code>train_model()</code> .
<code>horizons</code>	Optional. Filter results by horizon.
<code>windows</code>	Optional. Filter results by validation window number.
<code>facet_plot</code>	Adjust the plot display through <code>ggplot2::facet_grid()</code> . <code>facet_plot = NULL</code> plots results in one facet.
<code>group_filter</code>	Optional. A string for filtering plot results for grouped time-series (e.g., "group_col_1 == 'A'"); passed to <code>dplyr::filter()</code> internally.
<code>...</code>	Not used.

Value

Forecast plot of class 'ggplot'.

plot.lagged_df	<i>Plot datasets with lagged features</i>
----------------	---

Description

Plot datasets with lagged features to view their direct forecasting setup across horizons.

Usage

```
## S3 method for class 'lagged_df'
plot(x, ...)
```

Arguments

x	An object of class 'lagged_df' from create_lagged_df().
...	Not used.

Value

A single plot of class 'ggplot' if lookback was specified in create_lagged_df(); a list of plots, one per feature, of class 'ggplot' if lookback_control was specified.

Examples

```
# Sampled Seatbelts data from the R package datasets.
data("data_seatbelts", package = "forecastML")
#-----
# Example 1 - Training data for 3 horizon-specific models w/ common lags per predictor.
horizons <- c(1, 6, 12)
lookback <- 1:15

data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                              lookback = lookback, horizon = horizons)
plot(data_train)
#-----
# Example 2 - Training data for one 3-month horizon model w/ unique lags per predictor.
horizons <- 3
lookback <- list(c(3, 6, 9, 12), c(4:12), c(6:15), c(8))

data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                              lookback_control = lookback, horizon = horizons)
plot(data_train)
```

plot.training_results *Plot an object of class training_results*

Description

Several diagnostic plots can be returned to assess the quality of the forecasts based on predictions on the validation datasets.

Usage

```
## S3 method for class 'training_results'
plot(x, type = c("prediction", "residual",
  "forecast_stability", "forecast_variability"), models = NULL,
  horizons = NULL, windows = NULL, valid_indices = NULL,
  group_filter = NULL, ...)
```

Arguments

x	An object of class 'training_results' from predict.forecast_model().
type	Plot type. The default plot is "prediction" for validation dataset predictions.
models	Optional. Filter results by user-defined model name from train_model().
horizons	Optional. A numeric vector of model forecast horizons to filter results by horizon-specific model.
windows	Optional. A numeric vector of window numbers to filter results.
valid_indices	Optional. A numeric or date vector to filter results by validation row indices or dates.
group_filter	Optional. A string for filtering plot results for grouped time series (e.g., "group_col_1 == 'A'"). The results are passed to dplyr::filter() internally.
...	Not used.

Value

Diagnostic plots of class 'ggplot'.

plot.validation_error *Plot validation dataset forecast error*

Description

Plot forecast error at various levels of aggregation across validation datasets.

Usage

```
## S3 method for class 'validation_error'
plot(x, data_results, type = c("time",
  "horizon", "global"), models = NULL, horizons = NULL,
  windows = NULL, group_filter = NULL, ...)
```

Arguments

x	An object of class 'validation_error' from return_error().
data_results	An object of class 'training_results' from predict.forecast_model().
type	Select plot type; type = "time" is the default plot.
models	Optional. A vector of user-defined model names from train_model() to filter results.
horizons	Optional. A numeric vector to filter results by horizon.
windows	Optional. A numeric vector to filter results by validation window number.
group_filter	A string for filtering plot results for grouped time-series (e.g., "group_col_1 == 'A'").
...	Not used.

Value

Forecast error plots of class 'ggplot'.

plot.windows	<i>Plot validation datasets</i>
--------------	---------------------------------

Description

Plot validation datasets across time.

Usage

```
## S3 method for class 'windows'
plot(x, lagged_df, show_labels = TRUE,
  group_filter = NULL, ...)
```

Arguments

x	An object of class 'windows' from create_windows().
lagged_df	An object of class 'lagged_df' from create_lagged_df().
show_labels	Boolean. If TRUE, show validation dataset IDs on the plot.
group_filter	Optional. A string for filtering plot results for grouped time series (e.g., "group_col_1 == 'A'"). This string is passed to dplyr::filter() internally.
...	Not used.

Value

A plot of the outer-loop nested cross-validation windows of class 'ggplot'.

Examples

```
# Sampled Seatbelts data from the R package datasets.
data("data_seatbelts", package = "forecastML")

# Example - Training data for 3 horizon-specific models w/ common lags per predictor.
horizons <- c(1, 6, 12)
lookback <- 1:15

data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                              lookback = lookback, horizon = horizons)

# All historical window lengths of 12 plus any partial windows at the end of the dataset.
windows <- create_windows(data_train, window_length = 12)
plot(windows, data_train)

# Two custom validation windows with different lengths.
windows <- create_windows(data_train, window_start = c(20, 80), window_stop = c(30, 100))
plot(windows, data_train)
```

predict.forecast_model

Predict on validation datasets or forecast

Description

Predict with a 'forecast_model' object from train_model(). If data = create_lagged_df(..., type = "train"), predictions are returned for the outer-loop nested cross-validation datasets. If data is an object of class 'lagged_df' from create_lagged_df(..., type = "forecast"), predictions are returned for the horizons specified in create_lagged_df(horizons = ...).

Usage

```
## S3 method for class 'forecast_model'
predict(..., prediction_function = list(NULL),
        data)
```

Arguments

... One or more trained models from train_model().

prediction_function A list of user-defined prediction functions with length equal to the number of models supplied in The prediction functions take 2 required positional arguments—(1) a 'forecast_model' object from train_model() and (2) a data.frame

of model features from `create_lagged_df()`. For numeric outcomes, the function should return() 1- or 3-column data.frame of model predictions. If the prediction function returns a 1-column data.frame, point forecasts are assumed. If the prediction function returns a 3-column data.frame, lower and upper forecast bounds are assumed (the order and names of the 3 columns does not matter). For factor outcomes, the function should return() (1) 1-column data.frame of the model-predicted factor level or (2) an L-column data.frame of class probabilities where 'L' equals the number of levels in the outcome; columns should be ordered, from left to right, the same as `levels(data$outcome)` which is the default behavior for most `predict(..., type = "prob")` functions. Column names do not matter.

data If data is a training dataset from `create_lagged_df(..., type = "train")`, validation dataset predictions are returned; else, if data is a forecasting dataset from `create_lagged_df(..., type = "forecast")`, forecasts from horizons 1:h are returned.

Value

If data = `create_lagged_df(..., type = "forecast")`, an S3 object of class 'training_results'.
If data = `create_lagged_df(..., type = "forecast")`, an S3 object of class 'forecast_results'.

Columns in returned 'training_results' data.frame:

- **model**: User-supplied model name in `train_model()`.
- **model_forecast_horizon**: The direct-forecasting time horizon that the model was trained on.
- **window_length**: Validation window length measured in dataset rows.
- **window_number**: Validation dataset number.
- **valid_indices**: Validation dataset row names from `attributes(create_lagged_df())$row_indices`.
- **date_indices**: If given, validation dataset date indices from `attributes(create_lagged_df())$date_indices`.
- **"groups"**: If given, the user-supplied groups in `create_lagged_df()`.
- **"outcome_name"**: The target being forecasted.
- **"outcome_name"_pred**: The model predictions.
- **"outcome_name"_pred_lower**: If given, the lower prediction bounds returned by the user-supplied prediction function.
- **"outcome_name"_pred_upper**: If given, the upper prediction bounds returned by the user-supplied prediction function.

Columns in returned 'forecast_results' data.frame:

- **model**: User-supplied model name in `train_model()`.
- **model_forecast_horizon**: The direct-forecasting time horizon that the model was trained on.
- **horizon**: Forecast horizons, 1:h, measured in dataset rows.
- **window_length**: Validation window length measured in dataset rows.

- `forecast_period`: The forecast period in row indices or dates. The forecast period starts at either `attributes(create_lagged_df())$data_stop + 1` for row indices or `attributes(create_lagged_df())$data_stop + 1 * frequency` for date indices.
- `"groups"`: If given, the user-supplied groups in `create_lagged_df()`.
- `"outcome_name"`: The target being forecasted.
- `"outcome_name"_pred`: The model forecasts.
- `"outcome_name"_pred_lower`: If given, the lower forecast bounds returned by the user-supplied prediction function.
- `"outcome_name"_pred_upper`: If given, the upper forecast bounds returned by the user-supplied prediction function.

Examples

```
# Sampled Seatbelts data from the R package datasets.
data("data_seatbelts", package = "forecastML")

# Example - Training data for 2 horizon-specific models w/ common lags per predictor.
horizons <- c(1, 12)
lookback <- 1:15

data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                              lookback = lookback, horizon = horizons)

# One custom validation window at the end of the dataset.
windows <- create_windows(data_train, window_start = 181, window_stop = 192)

# User-define model - LASSO
# A user-defined wrapper function for model training that takes the following
# arguments: (1) a horizon-specific data.frame made with create_lagged_df(..., type = "train")
# (e.g., my_lagged_df$horizon_h) and, optionally, (2) any number of additional named arguments
# which are passed as '...' in train_model().
library(glmnet)
model_function <- function(data, my_outcome_col) {

  x <- data[, -(my_outcome_col), drop = FALSE]
  y <- data[, my_outcome_col, drop = FALSE]
  x <- as.matrix(x, ncol = ncol(x))
  y <- as.matrix(y, ncol = ncol(y))

  model <- glmnet::cv.glmnet(x, y, nfolds = 3)
  return(model)
}

# my_outcome_col = 1 is passed in ... but could have been defined in model_function().
model_results <- train_model(data_train, windows, model_name = "LASSO", model_function,
                             my_outcome_col = 1)

# User-defined prediction function - LASSO
# The predict() wrapper takes two positional arguments. First,
# the returned model from the user-defined modeling function (model_function() above).
```

```

# Second, a data.frame of predictors--identical to the datasets returned from
# create_lagged_df(..., type = "train"). The function can return a 1- or 3-column data.frame
# with either (a) point forecasts or (b) point forecasts plus lower and upper forecast
# bounds (column order and column names do not matter).
prediction_function <- function(model, data_features) {

  x <- as.matrix(data_features, ncol = ncol(data_features))

  data_pred <- data.frame("y_pred" = predict(model, x, s = "lambda.min"))
  return(data_pred)
}

# Predict on the validation datasets.
data_valid <- predict(model_results, prediction_function = list(prediction_function),
                      data = data_train)

# Forecast.
data_forecast <- create_lagged_df(data_seatbelts, type = "forecast", outcome_col = 1,
                                 lookback = lookback, horizon = horizons)

data_forecasts <- predict(model_results, prediction_function = list(prediction_function),
                          data = data_forecast)

```

return_error

Compute forecast error

Description

Compute forecast error metrics on the validation datasets or a new test dataset.

Usage

```

return_error(data_results, data_test = NULL, test_indices = NULL,
             metrics = c("mae", "mape", "mdape", "smape"), models = NULL,
             horizons = NULL, windows = NULL, group_filter = NULL)

```

Arguments

data_results	An object of class 'training_results' or 'forecast_results' from running (a) <code>predict</code> on a trained model or (b) <code>combine_forecasts()</code> .
data_test	Required for forecast results only. If data_results is an object of class 'forecast_results', a data.frame used to assess the accuracy of a 'forecast_results' object. data_test should have the outcome/target columns and any grouping columns.
test_indices	Required if data_test is given. A vector or 1-column data.frame of numeric row indices or dates (class 'Date' or 'POSIXt') with length nrow(data_test).
metrics	A character vector of common forecast error metrics. The default behavior is to return all metrics.

models	Optional. A character vector of user-defined model names supplied to <code>train_model()</code> .
horizons	Optional. A numeric vector to filter results by horizon.
windows	Optional. A numeric vector to filter results by validation window number.
group_filter	Optional. A string for filtering plot results for grouped time-series (e.g., "group_col_1 == 'A'"). <code>group_filter</code> is passed to <code>dplyr::filter()</code> internally.

Value

An S3 object of class 'validation_error' or 'forecast_error': A list of data.frames of error metrics for the validation datasets or forecast dataset depending on the `data_test` argument. An input to `data_results` from `combine_forecasts()` will return a single data.frame with results for each model passed in ...

A list containing:

- Error metrics by horizon + validation window
- Error metrics by horizon, collapsed across validation windows
- Global error metrics collapsed across horizons and validation windows

Error Metrics

- mae: Mean absolute error
- mape: Mean absolute percentage error
- mdape: Median absolute percentage error
- smape: Symmetrical mean absolute percentage error

Methods and related functions

The output of `return_error()` has the following generic S3 methods

- `plot` from `return_error()`

Examples

```
# Sampled Seatbelts data from the R package datasets.
data("data_seatbelts", package = "forecastML")

# Example - Training data for 2 horizon-specific models w/ common lags per predictor.
horizons <- c(1, 12)
lookback <- 1:15

data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                             lookback = lookback, horizon = horizons)

# One custom validation window at the end of the dataset.
windows <- create_windows(data_train, window_start = 181, window_stop = 192)
```



```

# User-define model - LASSO
# A user-defined wrapper function for model training that takes the following
# arguments: (1) a horizon-specific data.frame made with create_lagged_df(..., type = "train")
# (e.g., my_lagged_df$horizon_h) and, optionally, (2) any number of additional named arguments
# which are passed as '...' in train_model().
library(glmnet)
model_function <- function(data, my_outcome_col) {

  x <- data[, -(my_outcome_col), drop = FALSE]
  y <- data[, my_outcome_col, drop = FALSE]
  x <- as.matrix(x, ncol = ncol(x))
  y <- as.matrix(y, ncol = ncol(y))

  model <- glmnet::cv.glmnet(x, y, nfolds = 3)
  return(model)
}

# my_outcome_col = 1 is passed in ... but could have been defined in model_function().
model_results <- train_model(data_train, windows, model_name = "LASSO", model_function,
                             my_outcome_col = 1)

# User-defined prediction function - LASSO
# The predict() wrapper takes two positional arguments. First,
# the returned model from the user-defined modeling function (model_function() above).
# Second, a data.frame of predictors--identical to the datasets returned from
# create_lagged_df(..., type = "train"). The function can return a 1- or 3-column data.frame
# with either (a) point forecasts or (b) point forecasts plus lower and upper forecast
# bounds (column order and column names do not matter).
prediction_function <- function(model, data_features) {

  x <- as.matrix(data_features, ncol = ncol(data_features))

  data_pred <- data.frame("y_pred" = predict(model, x, s = "lambda.min"))
  return(data_pred)
}

# Predict on the validation datasets.
data_valid <- predict(model_results, prediction_function = list(prediction_function),
                     data = data_train)

# Forecast error metrics for validation datasets.
data_error <- return_error(data_valid)

```

return_hyper

Return model hyperparameters across validation datasets

Description

The purpose of this function is to support investigation into the stability of hyperparameters in the nested cross-validation and across forecast horizons.

Usage

```
return_hyper(forecast_model, hyper_function)
```

Arguments

`forecast_model` An object of class 'forecast_model' from [train_model](#).

`hyper_function` A user-defined function for retrieving model hyperparameters. See the example below for details.

Value

An S3 object of class 'forecast_model_hyper': A data.frame of model-specific hyperparameters.

Methods and related functions

The output of `return_hyper()` has the following generic S3 methods

- [plot](#)

Examples

```
# Sampled Seatbelts data from the R package datasets.
data("data_seatbelts", package = "forecastML")

# Example - Training data for 2 horizon-specific models w/ common lags per predictor.
horizons <- c(1, 12)
lookback <- 1:15

data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                              lookback = lookback, horizon = horizons)

# One custom validation window at the end of the dataset.
windows <- create_windows(data_train, window_start = 181, window_stop = 192)

# User-define model - LASSO
# A user-defined wrapper function for model training that takes the following
# arguments: (1) a horizon-specific data.frame made with create_lagged_df(..., type = "train")
# (e.g., my_lagged_df$horizon_h) and, optionally, (2) any number of additional named arguments
# which are passed as '...' in train_model().
library(glmnet)
model_function <- function(data, my_outcome_col) {

  x <- data[, -(my_outcome_col), drop = FALSE]
  y <- data[, my_outcome_col, drop = FALSE]
  x <- as.matrix(x, ncol = ncol(x))
  y <- as.matrix(y, ncol = ncol(y))

  model <- glmnet::cv.glmnet(x, y, nfolds = 3)
  return(model)
}
```

```

# my_outcome_col = 1 is passed in ... but could have been defined in model_function().
model_results <- train_model(data_train, windows, model_name = "LASSO", model_function,
                             my_outcome_col = 1)

# User-defined prediction function - LASSO
# The predict() wrapper takes two positional arguments. First,
# the returned model from the user-defined modeling function (model_function() above).
# Second, a data.frame of predictors--identical to the datasets returned from
# create_lagged_df(..., type = "train"). The function can return a 1- or 3-column data.frame
# with either (a) point forecasts or (b) point forecasts plus lower and upper forecast
# bounds (column order and column names do not matter).
prediction_function <- function(model, data_features) {

  x <- as.matrix(data_features, ncol = ncol(data_features))

  data_pred <- data.frame("y_pred" = predict(model, x, s = "lambda.min"))
  return(data_pred)
}

# Predict on the validation datasets.
data_valid <- predict(model_results, prediction_function = list(prediction_function),
                      data = data_train)

# User-defined hyperparameter function - LASSO
# The hyperparameter function should take one positional argument--the returned model
# from the user-defined modeling function (model_function() above). It should
# return a 1-row data.frame of the optimal hyperparameters.
hyper_function <- function(model) {

  lambda_min <- model$lambda.min
  lambda_1se <- model$lambda.1se

  data_hyper <- data.frame("lambda_min" = lambda_min, "lambda_1se" = lambda_1se)
  return(data_hyper)
}

data_error <- return_error(data_valid)

data_hyper <- return_hyper(model_results, hyper_function)

plot(data_hyper, data_valid, data_error, type = "stability", horizons = c(1, 12))

```

summary.lagged_df

Return a summary of a lagged_df object

Description

Return a summary of a lagged_df object

Usage

```
## S3 method for class 'lagged_df'
summary(object, ...)
```

Arguments

object An object of class 'lagged_df' from `create_lagged_df()`.
 ... Not used.

Value

A printed summary of the contents of the `lagged_df` object.

<code>train_model</code>	<i>Train a model across horizons and validation datasets</i>
--------------------------	--

Description

Train a user-defined forecast model for each horizon, `h`, and across the validation datasets, `d`. A total of $h * d$ models are trained—more if the user-defined modeling function performs any inner-loop cross-validation. These models can, however, be trained in parallel with the `future` package.

Usage

```
train_model(lagged_df, windows, model_name, model_function, ...,
            use_future = FALSE)
```

Arguments

`lagged_df` An object of class 'lagged_df' from [create_lagged_df](#).
`windows` An object of class 'windows' from [create_windows](#).
`model_name` A name for the model.
`model_function` A user-defined wrapper function for model training that takes the following arguments: (1) a horizon-specific `data.frame` made with `create_lagged_df(..., type = "train")` (i.e., the datasets stored in `lagged_df`) and, optionally, (2) any number of additional named arguments which can be passed in ... in this function.
 ... Optional. Named arguments passed into the user-defined `model_function`.
`use_future` Boolean. If `TRUE`, the `future` package is used for training models in parallel. The models will train in parallel across either (1) model forecast horizons or (2) validation windows, whichever is longer (i.e., `length(create_lagged_df())` or `nrow(create_windows())`). The user should run `future::plan(future::multiprocess)` or similar prior to this function to train these models in parallel.

Value

An S3 object of class 'forecast_model': A nested list of trained models. Models can be accessed with `my_trained_model$horizon_h$window_w$model` where 'h' gives the forecast horizon and 'w' gives the validation dataset window number from `create_windows()`.

Methods and related functions

The output of `train_model` can be passed into

- `return_error`
- `return_hyper`

and has the following generic S3 methods

- `predict`
- `plot` (from `predict.forecast_model(data = create_lagged_df(..., type = "train"))`)
- `plot` (from `predict.forecast_model(data = create_lagged_df(..., type = "forecast"))`)

Examples

```
# Sampled Seatbelts data from the R package datasets.
data("data_seatbelts", package = "forecastML")

# Example - Training data for 2 horizon-specific models w/ common lags per predictor.
horizons <- c(1, 12)
lookback <- 1:15

data_train <- create_lagged_df(data_seatbelts, type = "train", outcome_col = 1,
                              lookback = lookback, horizon = horizons)

# One custom validation window at the end of the dataset.
windows <- create_windows(data_train, window_start = 181, window_stop = 192)

# User-define model - LASSO
# A user-defined wrapper function for model training that takes the following
# arguments: (1) a horizon-specific data.frame made with create_lagged_df(..., type = "train")
# (e.g., my_lagged_df$horizon_h) and, optionally, (2) any number of additional named arguments
# which are passed as '...' in train_model().
library(glmnet)
model_function <- function(data, my_outcome_col) {

  x <- data[, -(my_outcome_col), drop = FALSE]
  y <- data[, my_outcome_col, drop = FALSE]
  x <- as.matrix(x, ncol = ncol(x))
  y <- as.matrix(y, ncol = ncol(y))

  model <- glmnet::cv.glmnet(x, y, nfolds = 3)
  return(model)
}

# my_outcome_col = 1 is passed in ... but could have been defined in model_function().
```

```
model_results <- train_model(data_train, windows, model_name = "LASSO", model_function,  
                             my_outcome_col = 1)  
  
# View the results for the model (a) trained on the first horizon  
# and (b) to be assessed on the first outer-loop validation window.  
model_results$horizon_1$window_1$model
```

Index

*Topic **datasets**

data_buoy, [9](#)

data_buoy_gaps, [10](#)

data_seatbelts, [11](#)

combine_forecasts, [2](#)

create_lagged_df, [4](#), [8](#), [11](#), [12](#), [28](#)

create_windows, [7](#), [8](#), [28](#)

data_buoy, [9](#), [10](#)

data_buoy_gaps, [10](#)

data_seatbelts, [11](#)

fill_gaps, [5](#), [11](#)

plot, [3](#), [7](#), [9](#), [24](#), [26](#), [29](#)

plot.forecast_model_hyper, [14](#)

plot.forecast_results, [16](#)

plot.forecastML, [13](#)

plot.lagged_df, [17](#)

plot.training_results, [18](#)

plot.validation_error, [18](#)

plot.windows, [19](#)

predict, [23](#), [29](#)

predict.forecast_model, [20](#)

return_error, [23](#), [29](#)

return_hyper, [25](#), [29](#)

summary, [7](#)

summary.lagged_df, [27](#)

train_model, [9](#), [11](#), [12](#), [26](#), [28](#)