Package 'tidysynth'

March 24, 2025

Title A Tidy Implementation of the Synthetic Control Method

Version 0.2.1

Description A synthetic control offers a way of evaluating the effect of an intervention in comparative case studies. The package makes a number of improvements when implementing the method in R. These improvements allow users to inspect, visualize, and tune the synthetic control more easily. A key benefit of a tidy implementation is that the entire preparation process for building the synthetic control can be accomplished in a single pipe.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

Depends R (>= 3.5.0)

Imports magrittr, tibble, dplyr, ggplot2, tidyr, forcats, rlang, kernlab, rgenoud, optimx, stats

Suggests testthat

NeedsCompilation no

Author Eric Dunford [aut, cre] (<https://orcid.org/0000-0003-3056-8687>)

Maintainer Eric Dunford <ed769@georgetown.edu>

Repository CRAN

Date/Publication 2025-03-24 00:50:02 UTC

Contents

`generate_control` *generate_control*

Description

Uses the weights generated from `generate_weights()` to weight control units from the donor pool to generate a synthetic version of the treated unit time series.

Usage

```
generate_control(data)
```

Arguments

`data`	nested data of type `tbl_df` generated from `synthetic_control()`. See `synthetic_control()` documentation for more information. In addition, `.unit_weights` must be generate using `generate_weights()`. See documentation for more information on how to generate weights.

Value

`tbl_df` with nested fields containing the following:

- `.id`: unit id for the intervention case (this will differ when a placebo unit).
- `.placebo`: indicator field taking on the value of 1 if a unit is a placebo unit, 0 if it's the specified treated unit.
- `.type`: type of the nested data construct: `treated` or `controls`. Keeps tract of which data construct is located in `.outcome` field.
- `.outcome`: nested data construct containing the outcome variable configured for the sythnetic control method. Data is configured into a wide format for the optimization task.
- `.predictors`: nested data construct containing the covariate matrices for the treated and control (donor) units. Data is configured into a wide format for the optimization task.

- `.synthetic_control`: nested data construct containing the synthetic control version of the outcome variable generated from the unit weights.
- `.unit_weights`: Nested column of unit weights (i.e. how each unit from the donor pool contributes to the synthetic control). Weights should sum to 1.
- `.predictor_weights`: Nested column of predictor variable weights (i.e. the significance of each predictor in optimizing the weights that generate the synthetic control). Weights should sum to 1. If variable weights are provided, those variable weights are provided.
- `.original_data`: original impute data filtered by treated or control units. This allows for easy processing down stream when generating predictors.
- `.meta`: stores information regarding the unit and time index, the treated unit and time and the name of the outcome variable. Used downstream in subsequent functions.
- `.loss`: the RMPE loss for both sets of weights.

Examples

```
# Smoking example data
data(smoking)

smoking_out <-
smoking %>%

# initial the synthetic control object
synthetic_control(outcome = cigsale,
                  unit = state,
                  time = year,
                  i_unit = "California",
                  i_time = 1988,
                  generate_placebos= FALSE) %>%

# Generate the aggregate predictors used to generate the weights
  generate_predictor(time_window=1980:1988,
                     lnincome = mean(lnincome, na.rm = TRUE),
                     retprice = mean(retprice, na.rm = TRUE),
                     age15to24 = mean(age15to24, na.rm = TRUE)) %>%

  generate_predictor(time_window=1984:1988,
                     beer = mean(beer, na.rm = TRUE)) %>%

  generate_predictor(time_window=1975,
                     cigsale_1975 = cigsale) %>%

  generate_predictor(time_window=1980,
                     cigsale_1980 = cigsale) %>%

  generate_predictor(time_window=1988,
                     cigsale_1988 = cigsale) %>%
```

```
  # Generate the fitted weights for the synthetic control
  generate_weights(optimization_window =1970:1988,
                   Margin.ipop=.02,Sigf.ipop=7,Bound.ipop=6) %>%

  # Generate the synthetic control
  generate_control()

# Plot the observed and synthetic trend
smoking_out %>% plot_trends(time_window = 1970:2000)
```

`generate_predictor` *generate_predictor*

Description

Create one or more scalar variables summarizing covariate data across a specified time window. These predictor variables are used to fit the synthetic control.

Usage

```
generate_predictor(data, time_window = NULL, ...)
```

Arguments

`data`	nested data of type `tbl_df` generated from `synthetic_control()`. See `synthetic_control()` documentation for more information.
`time_window`	set time window from the pre-intervention period that the data should be aggregated across to generate the specific predictor. Default is to use the entire pre-intervention period.
`...`	Name-value pairs of summary functions. The name will be the name of the variable in the result. The value should be an expression that returns a single value like min(x), n(), or sum(is.na(y)). Note that for all summary functions `na.rm = TRUE` argument should be specified as aggregating across units with missing values is a common occurrence.

Details

matrices of aggregate-level covariates to be used in the following minimization task.

$$W^*(V) = min \sum_{m=1}^{M} v_m (X_{1m} - \sum_{j=2}^{J+1} w_j X_{jm})^2$$

The importance of the generate predictors are determine by vector V, and the weights that determine unit-level importance are determined by vector W. The nested optimation task seeks to find optimal values of V and W. Note also that V can be provided by the user. See `?generate_weights()`.

Value

`tbl_df` with nested fields containing the following:

- `.id`: unit id for the intervention case (this will differ when a placebo unit).
- `.placebo`: indicator field taking on the value of 1 if a unit is a placebo unit, 0 if it's the specified treated unit.
- `.type`: type of the nested data construct: `treated` or `controls`. Keeps tract of which data construct is located in `.outcome` field.
- `.outcome`: nested data construct containing the outcome variable configured for the sythnetic control method. Data is configured into a wide format for the optimization task.
- `.predictors`: nested data construct containing the covariate matrices for the treated and control (donor) units. Data is configured into a wide format for the optimization task.
- `.original_data`: original impute data filtered by treated or control units. This allows for easy processing down stream when generating predictors.
- `.meta`: stores information regarding the unit and time index, the treated unit and time and the name of the outcome variable. Used downstream in subsequent functions.

Examples

```
# Smoking example data
data(smoking)

smoking_out <-
smoking %>%

# initial the synthetic control object
synthetic_control(outcome = cigsale,
                  unit = state,
                  time = year,
                  i_unit = "California",
                  i_time = 1988,
                  generate_placebos= FALSE) %>%

# Generate the aggregate predictors used to generate the weights
  generate_predictor(time_window=1980:1988,
                     lnincome = mean(lnincome, na.rm = TRUE),
                     retprice = mean(retprice, na.rm = TRUE),
                     age15to24 = mean(age15to24, na.rm = TRUE))

# Extract respective predictor matrices
smoking_out %>% grab_predictors(type = "treated")
smoking_out %>% grab_predictors(type = "controls")
```

generate_weights *generate_weights*

Description

Generates weights from the the aggregate-level predictors to generate the synthetic control. These weights determine which variable and which unit from the donor pool is important in generating the synthetic control.

Usage

```
generate_weights(
  data,
  optimization_window = NULL,
  custom_variable_weights = NULL,
  include_fit = FALSE,
  optimization_method = c("Nelder-Mead", "BFGS"),
  genoud = FALSE,
  quadopt = "ipop",
  margin_ipop = 5e-04,
  sigf_ipop = 5,
  bound_ipop = 10,
  verbose = FALSE,
  ...
)
```

Arguments

data	nested data of type `tbl_df` generated from `sythetic_control()`. See `synthetic_control()` documentation for more information. In addition, a matrix of predictors must be prespecified using the `generate_predictor()` function. See documentation for more information on how to generate a predictor function.
optimization_window	the temporal window of the pre-intervention outcome time series to be used in the optimization task. Default behavior uses the entire pre-intervention time period.
custom_variable_weights	a vector of provided weights that define a variable's importance in the optimization task. The weights are intended to reflect the users prior regarding the relative significance of each variable. Vector must sum to one. Note that the method is significantly faster when a custom variable weights are provided. Default behavior assumes no wieghts are provided and thus must be learned from the data.
include_fit	Boolean flag, if TRUE, then the optimization output is included in the outputted `tbl_df`.

`optimization_method`
: string vector that specifies the optimization algorithms to be used. Permissable values are all optimization algorithms that are currently implemented in the optimx function (see this function for details). This list currently includes c('Nelder-Mead', 'BFGS', 'CG', 'L-BFGS-B', 'nlm', 'nlminb', 'spg', and 'ucminf"). If multiple algorithms are specified, synth will run the optimization with all chosen algorithms and then return the result for the best performing method. Default is c('Nelder-Mead','BFGS'). As an additional possibility, the user can also specify 'All' which means that synth will run the results over all algorithms in optimx.

`genoud`
: Logical flag. If true, synth embarks on a two step optimization. In the first step, genoud, an optimization function that combines evolutionary algorithm methods with a derivative-based (quasi-Newton) method to solve difficult optimization problems, is used to obtain a solution. See genoud for details. In the second step, the genoud results are passed to the optimization algorithm(s) chosen in optimxmethod for a local optimization within the neighborhood of the genoud solution. This two step optimization procedure will require much more computing time, but may yield lower loss in cases where the search space is highly irregular.

`quadopt`
: string vector that specifies the routine for quadratic optimization over w weights. possible values are "ipop" and "LowRankQP" (see ipop and LowRankQP for details). default is 'ipop'

`margin_ipop`
: setting for ipop optimization routine: how close we get to the constrains (see ipop for details)

`sigf_ipop`
: setting for ipop optimization routine: Precision (default: 7 significant figures (see ipop for details)

`bound_ipop`
: setting for ipop optimization routine: Clipping bound for the variables (see ipop for details)

`verbose`
: Logical flag. If TRUE then intermediate results will be shown.

`...`
: Additional arguments to be passed to optimx and or genoud to adjust optimization.

Details

Optimization

The method completes the following nested minimization task:

$$W^*(V) = min \sum_{m=1}^{M} v_m (X_{1m} - \sum_{j=2}^{J+1} w_j X_{jm})^2$$

Where X_1 and X_0, which are matrices of aggregate-level covariates, are generated using the `generate_predictor()` function. V denotes the variable weights with M reflecting the total number of predictor variables. Thus, the optimal weights are a function of V.

The weights themselves are optimized via the following:

$$\sum_{t=1}^{T_0}(Y_{1t} - \sum_{j=2}^{J=1} w_j^*(V)Y_{jt})^2$$

where T_0 denotes the pre-intervention period (or a specific optimization window supplied by the argument `time_window`); J denotes the number of control units from the donor pool, where $j = 1$ reflects the treated unit.

Thus, the weights are selected in a manner that produces a synthetic $\hat{Y}$ that approximates the observed Y as closely as possible.

Variable Weights

As proposed in Abadie and Gardeazabal (2003) and Abadie, Diamond, Hainmueller (2010), the synth function routinely searches for the set of weights that generate the best fitting convex combination of the control units. In other words, the predictor weight matrix V (`custom_variable_weights`) is chosen among all positive definite diagonal matrices such that MSPE is minimized for the pre-intervention period. Instead of using this data-driven procedures to search for the best fitting synthetic control group, the user may supply their own weights using the `custom_variable_weights` argument. These weights reflect the user's subjective assessment of the predictive power of the variables generated by `generate_predictor()`.

When generating weights for the placebo cases, the variable weights used for the fit of the treated unit optimization. This ensures comparability between the placebo and treated fits. In addition, it greatly decreases processing time as the variable weights do not be learned for every placebo entry.

Value

`tbl_df` with nested fields containing the following:

- `.id`: unit id for the intervention case (this will differ when a placebo unit).
- `.placebo`: indicator field taking on the value of 1 if a unit is a placebo unit, 0 if it's the specified treated unit.
- `.type`: type of the nested data construct: `treated` or `controls`. Keeps tract of which data construct is located in `.outcome` field.
- `.outcome`: nested data construct containing the outcome variable configured for the sythnetic control method. Data is configured into a wide format for the optimization task.
- `.predictors`: nested data construct containing the covariate matrices for the treated and control (donor) units. Data is configured into a wide format for the optimization task.
- `.unit_weights`: Nested column of unit weights (i.e. how each unit from the donor pool contributes to the synthetic control). Weights should sum to 1.
- `.predictor_weights`: Nested column of predictor variable weights (i.e. the significance of each predictor in optimizing the weights that generate the synthetic control). Weights should sum to 1. If variable weights are provided, those variable weights are provided.
- `.original_data`: original impute data filtered by treated or control units. This allows for easy processing down stream when generating predictors.
- `.meta`: stores information regarding the unit and time index, the treated unit and time and the name of the outcome variable. Used downstream in subsequent functions.
- `.loss`: the RMPE loss for both sets of weights.

Examples

```
# Smoking example data
data(smoking)

smoking_out <-
smoking %>%

# initial the synthetic control object
synthetic_control(outcome = cigsale,
                  unit = state,
                  time = year,
                  i_unit = "California",
                  i_time = 1988,
                  generate_placebos= TRUE) %>%

# Generate the aggregate predictors used to generate the weights
  generate_predictor(time_window=1980:1988,
                     lnincome = mean(lnincome, na.rm = TRUE),
                     retprice = mean(retprice, na.rm = TRUE),
                     age15to24 = mean(age15to24, na.rm = TRUE)) %>%

  generate_predictor(time_window=1984:1988,
                     beer = mean(beer, na.rm = TRUE)) %>%

  generate_predictor(time_window=1975,
                     cigsale_1975 = cigsale) %>%

  generate_predictor(time_window=1980,
                     cigsale_1980 = cigsale) %>%

  generate_predictor(time_window=1988,
                     cigsale_1988 = cigsale) %>%


  # Generate the fitted weights for the synthetic control
  generate_weights(optimization_window =1970:1988,
                   Margin.ipop=.02,Sigf.ipop=7,Bound.ipop=6)

# Retrieve weights
smoking_out %>% grab_predictor_weights()
smoking_out %>% grab_unit_weights()

# Retrieve the placebo weights as well.
smoking_out %>% grab_predictor_weights(placebo= TRUE)
smoking_out %>% grab_unit_weights(placebo= TRUE)

# Plot the unit weights
smoking_out %>% plot_weights()
```

grab_balance_table *grab_balance_table*

Description

Compare the distributions of the aggregate-level predictors for the observed intervention unit, the synthetic control, and the donor pool average. Table helps user compare the the level of balance produced by the synthetic control.

Usage

```
grab_balance_table(data)
```

Arguments

data	nested data of type tbl_df

Value

tibble data frame containing balance statistics between the observed/synthetic unit and the donor pool for each variable used to fit the synthetic control.

Examples

```
data(smoking)
smoking_out <-
smoking %>%
synthetic_control(outcome = cigsale,
                  unit = state,
                  time = year,
                  i_unit = "California",
                  i_time = 1988,
                  generate_placebos=FALSE) %>%
  generate_predictor(time_window=1980:1988,
                     lnincome = mean(lnincome, na.rm = TRUE),
                     retprice = mean(retprice, na.rm = TRUE),
                     age15to24 = mean(age15to24, na.rm = TRUE)) %>%
  generate_predictor(time_window=1984:1988,
                     beer = mean(beer, na.rm = TRUE)) %>%
  generate_predictor(time_window=1975,
                     cigsale_1975 = cigsale) %>%
  generate_predictor(time_window=1980,
                     cigsale_1980 = cigsale) %>%
  generate_predictor(time_window=1988,
                     cigsale_1988 = cigsale) %>%
  generate_weights(optimization_window =1970:1988,
                   Margin.ipop=.02,Sigf.ipop=7,Bound.ipop=6) %>%
  generate_control()
```

```
smoking_out %>% grab_balance_table()
```

---

grab_loss *grab_loss*

Description

Extract the RMSE loss of the optimized weights from the synth pipeline.

Usage

```
grab_loss(data)
```

Arguments

data	nested data of type tbl_df

Value

tibble data frame

Examples

```
# Smoking example data
data(smoking)

smoking_out <-
smoking %>%

# initial the synthetic control object
synthetic_control(outcome = cigsale,
                  unit = state,
                  time = year,
                  i_unit = "California",
                  i_time = 1988,
                  generate_placebos=TRUE) %>%

# Generate the aggregate predictors used to generate the weights
  generate_predictor(time_window=1980:1988,
                     lnincome = mean(lnincome, na.rm = TRUE),
                     retprice = mean(retprice, na.rm = TRUE),
                     age15to24 = mean(age15to24, na.rm = TRUE)) %>%

  generate_predictor(time_window=1984:1988,
```

```
                    beer = mean(beer, na.rm = TRUE)) %>%

  generate_predictor(time_window=1975,
                    cigsale_1975 = cigsale) %>%

  generate_predictor(time_window=1980,
                    cigsale_1980 = cigsale) %>%

  generate_predictor(time_window=1988,
                    cigsale_1988 = cigsale) %>%


  # Generate the fitted weights for the synthetic control
  generate_weights(optimization_window =1970:1988,
                   Margin.ipop=.02,Sigf.ipop=7,Bound.ipop=6) %>%

  # Generate the synthetic control
  generate_control()

# grab the MSPE loss from the optimization of the weights.
smoking_out %>% grab_loss()
```

---

grab_outcome *grab_outcome*

Description

Extract a data frame containing the outcome variable from the synth pipline.

Usage

```
grab_outcome(data, type = "treated", placebo = FALSE)
```

Arguments

data	nested data of type tbl_df
type	string specifying which version of the data to extract: "treated" or "control". Default is "treated".
placebo	boolean flag; if TRUE placebo values are returned as well (if available). Default is FALSE.

Value

tibble data frame

Examples

```
# Smoking example data
data(smoking)

smoking_out <-
smoking %>%

# initial the synthetic control object
synthetic_control(outcome = cigsale,
                  unit = state,
                  time = year,
                  i_unit = "California",
                  i_time = 1988,
                  generate_placebos=FALSE) %>%

# Generate the aggregate predictors used to generate the weights
  generate_predictor(time_window=1980:1988,
                     lnincome = mean(lnincome, na.rm = TRUE),
                     retprice = mean(retprice, na.rm = TRUE),
                     age15to24 = mean(age15to24, na.rm = TRUE)) %>%

  generate_predictor(time_window=1984:1988,
                     beer = mean(beer, na.rm = TRUE)) %>%

  generate_predictor(time_window=1975,
                     cigsale_1975 = cigsale) %>%

  generate_predictor(time_window=1980,
                     cigsale_1980 = cigsale) %>%

  generate_predictor(time_window=1988,
                     cigsale_1988 = cigsale) %>%


  # Generate the fitted weights for the synthetic control
  generate_weights(optimization_window =1970:1988,
                   Margin.ipop=.02,Sigf.ipop=7,Bound.ipop=6) %>%

  # Generate the synthetic control
  generate_control()

# Grab outcome data frame for the treated unit
smoking_out %>% grab_outcome()

# Grab outcome data frame for control units
smoking_out %>% grab_outcome(type="controls")
```

grab_predictors *grab_predictors*

Description

Extract the aggregate-level covariates generated by `generate_predictor()` from the synth pipeline.

Usage

```
grab_predictors(data, type = "treated", placebo = FALSE)
```

Arguments

`data`	nested data of type `tbl_df`
`type`	string specifying which version of the data to extract: "treated" or "control". Default is "treated".
`placebo`	boolean flag; if TRUE placebo values are returned as well (if available). Default is FALSE.

Value

tibble data frame

Examples

```
# Smoking example data
data(smoking)

smoking_out <-
smoking %>%

# initial the synthetic control object
synthetic_control(outcome = cigsale,
                  unit = state,
                  time = year,
                  i_unit = "California",
                  i_time = 1988,
                  generate_placebos=FALSE) %>%

# Generate the aggregate predictors used to generate the weights
  generate_predictor(time_window=1980:1988,
                     lnincome = mean(lnincome, na.rm = TRUE),
                     retprice = mean(retprice, na.rm = TRUE),
                     age15to24 = mean(age15to24, na.rm = TRUE)) %>%

  generate_predictor(time_window=1984:1988,
                     beer = mean(beer, na.rm = TRUE)) %>%
```

```
  generate_predictor(time_window=1975,
                     cigsale_1975 = cigsale) %>%

  generate_predictor(time_window=1980,
                     cigsale_1980 = cigsale) %>%

  generate_predictor(time_window=1988,
                     cigsale_1988 = cigsale) %>%


  # Generate the fitted weights for the synthetic control
  generate_weights(optimization_window =1970:1988,
                   Margin.ipop=.02,Sigf.ipop=7,Bound.ipop=6) %>%

  # Generate the synthetic control
  generate_control()

# Grab predictors data frame for the treated unit
smoking_out %>% grab_predictors()

# Grab predictors data frame for control units
smoking_out %>% grab_predictors(type="controls")
```

`grab_predictor_weights` *grab_predictor_weights*

Description

Extract the predictor variable weights generated by `generate_weights()` from the synth pipeline.

Usage

```
grab_predictor_weights(data, placebo = FALSE)
```

Arguments

`data`	nested data of type `tbl_df`
`placebo`	boolean flag; if TRUE placebo values are returned as well (if available). Default is FALSE.

Value

tibble data frame

Examples

```
# Smoking example data
data(smoking)

smoking_out <-
smoking %>%

# initial the synthetic control object
synthetic_control(outcome = cigsale,
                  unit = state,
                  time = year,
                  i_unit = "California",
                  i_time = 1988,
                  generate_placebos=TRUE) %>%

# Generate the aggregate predictors used to generate the weights
  generate_predictor(time_window=1980:1988,
                     lnincome = mean(lnincome, na.rm = TRUE),
                     retprice = mean(retprice, na.rm = TRUE),
                     age15to24 = mean(age15to24, na.rm = TRUE)) %>%

  generate_predictor(time_window=1984:1988,
                     beer = mean(beer, na.rm = TRUE)) %>%

  generate_predictor(time_window=1975,
                     cigsale_1975 = cigsale) %>%

  generate_predictor(time_window=1980,
                     cigsale_1980 = cigsale) %>%

  generate_predictor(time_window=1988,
                     cigsale_1988 = cigsale) %>%


  # Generate the fitted weights for the synthetic control
  generate_weights(optimization_window =1970:1988,
                   Margin.ipop=.02,Sigf.ipop=7,Bound.ipop=6) %>%

  # Generate the synthetic control
  generate_control()

# Grab the predictor weights data frame for the treated unit.
smoking_out %>% grab_predictor_weights()

# Grab the predictor weights data frame for the placebo units as well.
smoking_out %>% grab_predictor_weights(placebo=TRUE)
```

grab_significance *grab_significance*

Description

Generate inferential statistics comparing the rarety of the unit that actually received the intervention to the placebo units in the donor pool.

Usage

```
grab_significance(data, time_window = NULL)
```

Arguments

data	nested data of type tbl_df
time_window	time window that the significance values should be computed.

Details

Inferential statitics are generated by comparing the observed difference between the actual treated unit and its synthetic control to each placebo unit and its synthetic control. The rarity of the actual to the placebo is used to infer the likelihood of observing the effect.

Inference in this framework leverages the mean squared predictive error (MSPE) of the fit in the pre-period to the fit in the post-period as a ratio.

$$\frac{RMSE_{Post}}{RMSE_{Pre}}$$

The ratio captures the differences between the pre-intervention fit and the post-intervention divergence of the trend (i.e. the causal quantity). A good fit in the pre-period denotes that the observed and synthetic case tracked well together. Divergence in the post-period captures the difference brought about by the intervention in the two trends. Thus, when the ratio is high, we observe more of a difference between the two trends. If, however, the pre-period fit is poor, or there is not substantial divergence in the post-period, then this ratio amount will be smaller.

The Fisher's Exact P-Value is generated by ranking the ratios for the treated and placebo units. The P-Value is then calculated by dividing the rank of the case over the total (rank/total). The case with the highest RMSE ratio is rare given the distribution of cases as generated by the placebo. A more detailed outline of inference within the synthetic control framework can be found in Adabie et al. 2010.

Note that conventional significance levels are not achievable if there is an insufficient number of control cases. One needs at least 20 control case to use the conventional .05 level. With fewer cases, significance levels need to be adjusted to accommodate the low total rank. This is a bug of rank based significance metrics.

In addition to the Fisher's Precise P-Value, a Z-score is also included, which is just the standardized RMSE ratios for all the cases. The Z-Score captures the degree to which a particular case's RMSE ratio deviates from the distribution of the placebo cases.

Value

tibble data frame containing the following fields:

- `unit_name`: name of the unit
- `type`: treated or donor unit (placebo)
- `pre_mspe`: pre-intervention period means squared predictive error
- `post_mspe`: post-intervention period means squared predictive error
- `mspe_ratio`: post_mspe/pre_mspe; captures the difference in fit in the pre and post period. A good fit in the pre-period and a poor fit in the post-period reflects a meaningful effect when comparing the difference between the observed outcome and the synthetic control.
- `rank`: rank order of the mspe_ratio.
- `fishers_exact_pvalue`: rank/total to generate a p-value. Conventional levels aren't achievable if there isn't a sufficient number of controls to generate a large enough ranking. Need at least 20 control units to use the conventional .05 level.
- `z_score`: (mspe_ratio-mean(mspe_ratio))/sd(mspe_ratio); captures the degree to which the mspe_ratio of the treated unit deviates from the mean of the placebo units. Provinding an alternative significance determination.

Examples

```
# Smoking example data
data(smoking)

smoking_out <-
smoking %>%

# initial the synthetic control object
synthetic_control(outcome = cigsale,
                  unit = state,
                  time = year,
                  i_unit = "California",
                  i_time = 1988,
                  generate_placebos=FALSE) %>%

# Generate the aggregate predictors used to generate the weights
  generate_predictor(time_window=1980:1988,
                     lnincome = mean(lnincome, na.rm = TRUE),
                     retprice = mean(retprice, na.rm = TRUE),
                     age15to24 = mean(age15to24, na.rm = TRUE)) %>%

  generate_predictor(time_window=1984:1988,
                     beer = mean(beer, na.rm = TRUE)) %>%

  generate_predictor(time_window=1975,
                     cigsale_1975 = cigsale) %>%

  generate_predictor(time_window=1980,
```

```
                    cigsale_1980 = cigsale) %>%

  generate_predictor(time_window=1988,
                    cigsale_1988 = cigsale) %>%


  # Generate the fitted weights for the synthetic control
  generate_weights(optimization_window =1970:1988,
                  Margin.ipop=.02,Sigf.ipop=7,Bound.ipop=6) %>%

  # Generate the synthetic control
  generate_control()

# Plot the observed and synthetic trend
smoking_out %>% grab_significance(time_window = 1970:2000)
```

`grab_synthetic_control` *grab_synthetic_control*

Description

Extract the synthetic control as a data frame generated using `generate_control()` from the synth pipeline.

Usage

```
grab_synthetic_control(data, placebo = FALSE)
```

Arguments

`data`	nested data of type `tbl_df`
`placebo`	boolean flag; if TRUE placebo values are returned as well (if available). Default is FALSE.

Value

tibble data frame

Examples

```
# Smoking example data
data(smoking)
```

```
smoking_out <-
smoking %>%

# initial the synthetic control object
synthetic_control(outcome = cigsale,
                  unit = state,
                  time = year,
                  i_unit = "California",
                  i_time = 1988,
                  generate_placebos=TRUE) %>%

# Generate the aggregate predictors used to generate the weights
  generate_predictor(time_window=1980:1988,
                     lnincome = mean(lnincome, na.rm = TRUE),
                     retprice = mean(retprice, na.rm = TRUE),
                     age15to24 = mean(age15to24, na.rm = TRUE)) %>%

  generate_predictor(time_window=1984:1988,
                     beer = mean(beer, na.rm = TRUE)) %>%

  generate_predictor(time_window=1975,
                     cigsale_1975 = cigsale) %>%

  generate_predictor(time_window=1980,
                     cigsale_1980 = cigsale) %>%

  generate_predictor(time_window=1988,
                     cigsale_1988 = cigsale) %>%


  # Generate the fitted weights for the synthetic control
  generate_weights(optimization_window =1970:1988,
                   Margin.ipop=.02,Sigf.ipop=7,Bound.ipop=6) %>%

  # Generate the synthetic control
  generate_control()

# Grab a data frame containing the observed outcome and the synthetic control outcome
smoking_out %>% grab_synthetic_control()


# Grab the data frame with the placebos.
smoking_out %>% grab_synthetic_control(placebo=TRUE)
```

---

grab_unit_weights *grab_unit_weights*

Description

Extract the unit weights generated by `generate_weights()` from the synth pipeline.

Usage

```
grab_unit_weights(data, placebo = FALSE)
```

Arguments

`data`	nested data of type `tbl_df`
`placebo`	boolean flag; if TRUE placebo values are returned as well (if available). Default is FALSE.

Value

tibble data frame

Examples

```
# Smoking example data
data(smoking)

smoking_out <-
smoking %>%

# initial the synthetic control object
synthetic_control(outcome = cigsale,
                  unit = state,
                  time = year,
                  i_unit = "California",
                  i_time = 1988,
                  generate_placebos=TRUE) %>%

# Generate the aggregate predictors used to generate the weights
  generate_predictor(time_window=1980:1988,
                     lnincome = mean(lnincome, na.rm = TRUE),
                     retprice = mean(retprice, na.rm = TRUE),
                     age15to24 = mean(age15to24, na.rm = TRUE)) %>%

  generate_predictor(time_window=1984:1988,
                     beer = mean(beer, na.rm = TRUE)) %>%

  generate_predictor(time_window=1975,
                     cigsale_1975 = cigsale) %>%

  generate_predictor(time_window=1980,
                     cigsale_1980 = cigsale) %>%

  generate_predictor(time_window=1988,
```

```
                    cigsale_1988 = cigsale) %>%


  # Generate the fitted weights for the synthetic control
  generate_weights(optimization_window =1970:1988,
                   Margin.ipop=.02,Sigf.ipop=7,Bound.ipop=6)

# Grab the unit weights for the treated unit.
smoking_out %>% grab_unit_weights()

# Grab the unit weights for the placebo units as well.
smoking_out %>% grab_unit_weights(placebo=TRUE)
```

---

plot_differences *plot_difference*

Description

Plot the difference between the observed and synthetic control unit. The difference captures the causal quantity (i.e. the magnitude of the difference between the observed and counter-factual case).

Usage

```
plot_differences(data, time_window = NULL)
```

Arguments

data	nested data of type tbl_df.
time_window	time window of the trend plot.

Value

ggplot object of the difference between the observed and synthetic trends.

ggplot object of difference between the observed and synthetic control unit.

Examples

```
# Smoking example data
data(smoking)

smoking_out <-
smoking %>%
```

```
# initial the synthetic control object
synthetic_control(outcome = cigsale,
                  unit = state,
                  time = year,
                  i_unit = "California",
                  i_time = 1988,
                  generate_placebos=TRUE) %>%

# Generate the aggregate predictors used to generate the weights
  generate_predictor(time_window=1980:1988,
                     lnincome = mean(lnincome, na.rm = TRUE),
                     retprice = mean(retprice, na.rm = TRUE),
                     age15to24 = mean(age15to24, na.rm = TRUE)) %>%

  generate_predictor(time_window=1984:1988,
                     beer = mean(beer, na.rm = TRUE)) %>%

  generate_predictor(time_window=1975,
                     cigsale_1975 = cigsale) %>%

  generate_predictor(time_window=1980,
                     cigsale_1980 = cigsale) %>%

  generate_predictor(time_window=1988,
                     cigsale_1988 = cigsale) %>%


  # Generate the fitted weights for the synthetic control
  generate_weights(optimization_window =1970:1988,
                   Margin.ipop=.02,Sigf.ipop=7,Bound.ipop=6) %>%

  # Generate the synthetic control
  generate_control()

# Plot the observed and synthetic trend
smoking_out %>% plot_differences(time_window = 1970:2000)
```

---

plot_mspe_ratio *plot_mspe_ratio*

Description

Plot the MSPE ratios for each case (observed and placebos). The ratio is used for inference in the synthetic control setup. The following plot ranks the RMSE ratio's in descending order.

Usage

```
plot_mspe_ratio(data, time_window = NULL)
```

Arguments

`data`	nested data of type `tbl_df`.
`time_window`	time window that the pre- and post-period values should be used to compute the MSPE ratio.

Details

Inferential statitics are generated by comparing the observed difference between the actual treated unit and its synthetic control to each placebo unit and its synthetic control. The rarity of the actual to the placebo is used to infer the likelihood of observing the effect.

Inference in this framework leverages the mean squared predictive error (MSPE) of the fit in the pre-period to the fit in the post-period as a ratio.

$$\frac{RMSE_{Post}}{RMSE_{Pre}}$$

The ratio captures the differences between the pre-intervention fit and the post-intervention divergence of the trend (i.e. the causal quantity). A good fit in the pre-period denotes that the observed and synthetic case tracked well together. Divergence in the post-period captures the difference brought about by the intervention in the two trends. Thus, when the ratio is high, we observe more of a difference between the two trends. If, however, the pre-period fit is poor, or there is not substantial divergence in the post-period, then this ratio amount will be smaller. A more detailed outline of inference within the synthetic control framework can be found in Adabie et al. 2010.

Value

`ggplot` object plotting the MSPE ratios by case.

Examples

```
# Smoking example data
data(smoking)

smoking_out <-
smoking %>%

# initial the synthetic control object
synthetic_control(outcome = cigsale,
                  unit = state,
                  time = year,
                  i_unit = "California",
                  i_time = 1988,
                  generate_placebos=TRUE) %>%
```

```
  # Generate the aggregate predictors used to generate the weights
    generate_predictor(time_window=1980:1988,
                       lnincome = mean(lnincome, na.rm = TRUE),
                       retprice = mean(retprice, na.rm = TRUE),
                       age15to24 = mean(age15to24, na.rm = TRUE)) %>%

    generate_predictor(time_window=1984:1988,
                       beer = mean(beer, na.rm = TRUE)) %>%

    generate_predictor(time_window=1975,
                       cigsale_1975 = cigsale) %>%

    generate_predictor(time_window=1980,
                       cigsale_1980 = cigsale) %>%

    generate_predictor(time_window=1988,
                       cigsale_1988 = cigsale) %>%


    # Generate the fitted weights for the synthetic control
    generate_weights(optimization_window =1970:1988,
                     Margin.ipop=.02,Sigf.ipop=7,Bound.ipop=6) %>%

    # Generate the synthetic control
    generate_control()

  # Plot the observed and synthetic trend
  smoking_out %>% plot_mspe_ratio(time_window = 1970:2000)
```

`plot_placebos` *plot_placebos*

Description

Plot the difference between the observed and sythetic control unit for the treated and the placebo units. The difference captures the causal quantity (i.e. the magnitude of the difference between the observed and counterfactual case). Plotting the actual treated observation against the placebos captures the likelihood (or rarity) of the observed differenced trend.

Usage

```
plot_placebos(data, time_window = NULL, prune = TRUE)
```

Arguments

`data`	nested data of type `tbl_df`.

time_window	time window of the tbl_df plot.
prune	boolean flag; if TRUE, then all placebo cases with a pre-period RMSPE exceeding two times the treated unit pre-period RMSPE are pruned; Default is TRUE.

Details

The function provides a pruning rule where all placebo cases with a pre-period root mean squared predictive error (RMSPE) exceeding two times the treated unit pre-period RMSPE are pruned. This helps overcome scale issues when a particular placebo case has poor fit in the pre-period.

See documentation on ?synthetic_control on how to generate placebo cases. When initializing a synth pipeline, set the generate_placebos argument to TRUE. The processing pipeline remains the same.

Value

ggplot object of the difference between the observed and synthetic trends for the treated and placebo units.

Examples

```
# Smoking example data
data(smoking)

smoking_out <-
smoking %>%

# initial the synthetic control object
synthetic_control(outcome = cigsale,
                  unit = state,
                  time = year,
                  i_unit = "California",
                  i_time = 1988,
                  generate_placebos=TRUE) %>%

# Generate the aggregate predictors used to generate the weights
  generate_predictor(time_window=1980:1988,
                     lnincome = mean(lnincome, na.rm = TRUE),
                     retprice = mean(retprice, na.rm = TRUE),
                     age15to24 = mean(age15to24, na.rm = TRUE)) %>%

  generate_predictor(time_window=1984:1988,
                     beer = mean(beer, na.rm = TRUE)) %>%

  generate_predictor(time_window=1975,
                     cigsale_1975 = cigsale) %>%

  generate_predictor(time_window=1980,
                     cigsale_1980 = cigsale) %>%
```

```
  generate_predictor(time_window=1988,
                     cigsale_1988 = cigsale) %>%


  # Generate the fitted weights for the synthetic control
  generate_weights(optimization_window =1970:1988,
                   Margin.ipop=.02,Sigf.ipop=7,Bound.ipop=6) %>%

  # Generate the synthetic control
  generate_control()

# Plot the observed and synthetic trend
smoking_out %>% plot_placebos(time_window = 1970:2000)
```

`plot_trends` *plot_trends*

Description

Plot the observed and synthetic trends for the treated units.

Usage

```
plot_trends(data, time_window = NULL)
```

Arguments

`data`	nested data of type `tbl_df`.
`time_window`	time window of the trend plot.

Details

Synthetic control is a visual-based method, like Regression Discontinuity, so inspection of the pre-intervention period fits is key assessing the sythetic control's fit. A poor fit in the pre-period reduces confidence in the post-period trend capturing the counterfactual.

See `?generate_control()` for information on how to generate a synthetic control unit.

Value

`ggplot` object of the observed and synthetic trends.

Examples

```
# Smoking example data
data(smoking)

smoking_out <-
smoking %>%

# initial the synthetic control object
synthetic_control(outcome = cigsale,
                  unit = state,
                  time = year,
                  i_unit = "California",
                  i_time = 1988,
                  generate_placebos=TRUE) %>%

# Generate the aggregate predictors used to generate the weights
  generate_predictor(time_window=1980:1988,
                     lnincome = mean(lnincome, na.rm = TRUE),
                     retprice = mean(retprice, na.rm = TRUE),
                     age15to24 = mean(age15to24, na.rm = TRUE)) %>%

  generate_predictor(time_window=1984:1988,
                     beer = mean(beer, na.rm = TRUE)) %>%

  generate_predictor(time_window=1975,
                     cigsale_1975 = cigsale) %>%

  generate_predictor(time_window=1980,
                     cigsale_1980 = cigsale) %>%

  generate_predictor(time_window=1988,
                     cigsale_1988 = cigsale) %>%


  # Generate the fitted weights for the synthetic control
  generate_weights(optimization_window =1970:1988,
                   Margin.ipop=.02,Sigf.ipop=7,Bound.ipop=6) %>%

  # Generate the synthetic control
  generate_control()

# Plot the observed and synthetic trend
smoking_out %>% plot_trends(time_window = 1970:2000)
```

plot_weights *plot_weights*

Description

Plot the unit and predictor variable weights generated using generate_weights()

Usage

```
plot_weights(data)
```

Arguments

data	nested data of type tbl_df.

Details

See grab_unit_weights() and grab_predictor_weights()

Value

a ggplot object that plots the unit and variable weights.

Examples

```
# Smoking example data
data(smoking)

smoking_out <-
smoking %>%

# initial the synthetic control object
synthetic_control(outcome = cigsale,
                  unit = state,
                  time = year,
                  i_unit = "California",
                  i_time = 1988,
                  generate_placebos=TRUE) %>%

# Generate the aggregate predictors used to generate the weights
  generate_predictor(time_window=1980:1988,
                     lnincome = mean(lnincome, na.rm = TRUE),
                     retprice = mean(retprice, na.rm = TRUE),
                     age15to24 = mean(age15to24, na.rm = TRUE)) %>%

  generate_predictor(time_window=1984:1988,
                     beer = mean(beer, na.rm = TRUE)) %>%
```

```
  generate_predictor(time_window=1975,
                     cigsale_1975 = cigsale) %>%

  generate_predictor(time_window=1980,
                     cigsale_1980 = cigsale) %>%

  generate_predictor(time_window=1988,
                     cigsale_1988 = cigsale) %>%


  # Generate the fitted weights for the synthetic control
  generate_weights(optimization_window =1970:1988,
                   Margin.ipop=.02,Sigf.ipop=7,Bound.ipop=6) %>%

  # Generate the synthetic control
  generate_control()

# Plot the observed and synthetic trend
smoking_out %>% plot_weights()
```

`smoking` *smoking dataset*

Description

A dataset on the implementation of Proposition 99 in California in 1988. Data contains information on California and 38 other (control/donor) states used in Abadie et al. 2010's paper walking through the synthetic control method. Covers the time range 1970 to 2000

Usage

```
data(smoking)
```

Format

A data frame with 1209 rows and 7 variables:

state name of U.S. state

year year

cigsale cigarette sales pack per 100,000 people

lnincome log mean income

beer beer sales per 100,000 people

age15to24 Proportion of the population between 15 and 24

retprice Retail price of a box of cigarettes

Source

https://economics.mit.edu/files/11859

References

Abadie, A., Diamond, A. and Hainmueller, J., 2010. Synthetic control methods for comparative case studies: Estimating the effect of California's tobacco control program. Journal of the American statistical Association, 105(490), pp.493-505.

`synthetic_control` *synthetic_control*

Description

`synthetic_control()` declares the input data frame for use in the synthetic control method. Allows for the specification of the panel units along with the intervention unit and time (`treated`). All units that are not the designated treated units are entered into the donor pool from which the synthetic control is generated. All time points prior and equal to the intervention time are designated as the pre-intervention period; and all time periods after are the post-intervention period.

Usage

```
synthetic_control(
  data = NULL,
  outcome = NULL,
  unit = NULL,
  time = NULL,
  i_unit = NULL,
  i_time = NULL,
  generate_placebos = TRUE
)
```

Arguments

`data`	panel data frame in long format (i.e. unit of analysis is unit-time period, such as country-year) containing both treated and control donor pool units. All units/time periods that are not desired to be in the donor should be excluded prior to passing to `synthetic_control()`.
`outcome`	Name of the outcome variable. Outcome variable should be a continuous measure that is observed across multiple time points.
`unit`	Name of the case unit variable in the panel data.
`time`	Name of the time unit variable in the panel data.
`i_unit`	Name of the treated case unit where the intervention occurred.
`i_time`	Name of the treated time period when the intervention occurred.

generate_placebos
: logical flag requesting that placebo versions of the data be generated for downstream inferential methods. Generates a version of the nested data where each control unit is the intervention unit. Default is TRUE.

Details

Note that `synthetic_control()` also allows for the simultaneous generation of placebo units (i.e. units where the treated unit is one of the controls). The addition of the placebo units increases computation time (as a synthetic control needs to be generated for each placebo unit) but it allows for inference as outlined in Abadie et al. 2010.

Value

`tbl_df` with nested fields containing the following:

- `.id`: unit id for the intervention case (this will differ when a placebo unit).
- `.placebo`: indicator field taking on the value of 1 if a unit is a placebo unit, 0 if it's the specified treated unit.
- `.type`: type of the nested data construct: `treated` or `controls`. Keeps tract of which data construct is located in `.outcome` field.
- `.outcome`: nested data construct containing the outcome variable configured for the sythnetic control method. Data is configured into a wide formate for the optimization task.
- `.original_data`: original impute data filtered by treated or control units. This allows for easy processing down stream when generating predictors.
- `.meta`: stores information regarding the unit and time index, the treated unit and time and the name of the outcome variable. Used downstream in subsequent functions.

Examples

```
############################
###### Basic Example #######
############################


# Smoking example data
data(smoking)

# initial the synthetic control object
smoking_out <-
smoking %>%
synthetic_control(outcome = cigsale,
                  unit = state,
                  time = year,
                  i_unit = "California",
                  i_time = 1988,
                  generate_placebos= FALSE)

# data configuration
dplyr::glimpse(smoking_out)
```

```
# Grap the organized outcome variables
smoking_out %>% grab_outcome(type = "treated")
smoking_out %>% grab_outcome(type = "controls")


###################################
####### Full implementation #######
###################################


# Smoking example data
data(smoking)

smoking_out <-
smoking %>%

# initial the synthetic control object
synthetic_control(outcome = cigsale,
                  unit = state,
                  time = year,
                  i_unit = "California",
                  i_time = 1988,
                  generate_placebos= FALSE) %>%

# Generate the aggregate predictors used to generate the weights
  generate_predictor(time_window=1980:1988,
                     lnincome = mean(lnincome, na.rm = TRUE),
                     retprice = mean(retprice, na.rm = TRUE),
                     age15to24 = mean(age15to24, na.rm = TRUE)) %>%

  generate_predictor(time_window=1984:1988,
                     beer = mean(beer, na.rm = TRUE)) %>%

  generate_predictor(time_window=1975,
                     cigsale_1975 = cigsale) %>%

  generate_predictor(time_window=1980,
                     cigsale_1980 = cigsale) %>%

  generate_predictor(time_window=1988,
                     cigsale_1988 = cigsale) %>%


  # Generate the fitted weights for the synthetic control
  generate_weights(optimization_window =1970:1988,
                   Margin.ipop=.02,Sigf.ipop=7,Bound.ipop=6) %>%

  # Generate the synthetic control
  generate_control()

# Plot the observed and synthetic trend
smoking_out %>% plot_trends(time_window = 1970:2000)
```

synth_method *synth_method*

Description

AUX Function: Original synthetic control method proposed by (Abadie et al. 2003, 2010, 2015) and implemented in synth package. Method has been commandeered for internal use here.

Usage

```
synth_method(
  treatment_unit_covariates = NULL,
  control_units_covariates = NULL,
  control_units_outcome = NULL,
  treatment_unit_outcome = NULL,
  custom.v = NULL,
  optimxmethod = c("Nelder-Mead", "BFGS"),
  genoud = FALSE,
  Margin.ipop = 5e-04,
  Sigf.ipop = 5,
  Bound.ipop = 10,
  verbose = FALSE,
  ...
)
```

Arguments

treatment_unit_covariates	matrix of treated predictor data
control_units_covariates	matrix of controls' predictor data.
control_units_outcome	matrix of controls' outcome data for the pre-treatment periods over which MSPE is to be minimized.
treatment_unit_outcome	matrix of treated outcome data for the pre-treatment periods over which MSPE is to be minimized.
custom.v	vector of weights for predictors supplied by the user. uses synth to bypass optimization for solution.V. See details.

optimxmethod	string vector that specifies the optimization algorithms to be used. Permissible values are all optimization algorithms that are currently implemented in the optimx function (see this function for details). This list currently includes c("Nelder-Mead', 'BFGS', 'CG', 'L-BFGS-B', 'nlm', 'nlminb', 'spg', and 'ucminf"). If multiple algorithms are specified, synth will run the optimization with all chosen algorithms and then return the result for the best performing method. Default is c("Nelder-Mead", "BFGS"). As an additional possibility, the user can also specify 'All' which means that synth will run the results over all algorithms in optimx.
genoud	Logical flag. If true, synth embarks on a two step optimization. In the first step, genoud, an optimization function that combines evolutionary algorithm methods with a derivative-based (quasi-Newton) method to solve difficult optimization problems, is used to obtain a solution. See genoud for details. In the second step, the genoud results are passed to the optimization algorithm(s) chosen in optimxmethod for a local optimization within the neighborhood of the genoud solution. This two step optimization procedure will require much more computing time, but may yield lower loss in cases where the search space is highly irregular.
Margin.ipop	setting for ipop optimization routine: how close we get to the constrains (see ipop for details)
Sigf.ipop	setting for ipop optimization routine: Precision (default: 7 significant figures (see ipop for details)
Bound.ipop	setting for ipop optimization routine: Clipping bound for the variables (see ipop for details)
verbose	Logical flag. If TRUE then intermediate results will be shown.
...	Additional arguments to be passed to optimx and or genoud to adjust optimization.

Details

Synth works as the main engine of the tidysynth package. More on the method and estimation procedures can be found in (Abadie et al. 2010).

As proposed in Abadie and Gardeazabal (2003) and Abadie, Diamond, Hainmueller (2010), the synth function routinely searches for the set of weights that generate the best fitting convex combination of the control units. In other words, the predictor weight matrix V is chosen among all positive definite diagonal matrices such that MSPE is minimized for the pre-intervention period. Instead of using this data-driven procedures to search for the best fitting synthetic control group, the user may supply his own vector of V weights, based on his subjective assessment of the predictive power of the variables in treatment_unit_covariates and control_units_covariates. In this case, the vector of V weights for each variable should be supplied via the custom.V option in synth and the optimization over the V matrices is bypassed.

Value

solution.v = vector of predictor weights; solution.w = vector of weights across the controls; loss.v = MSPE from optimization over v and w weights; loss.w = Loss from optimization over w weights;

custom.v =if this argument was specified in the call to synth, this outputs the weight vector specified; rgV.optim = Results from optimx() minimization. Could be used for diagnostics.

`synth_weights` *synth_weights*

Description

Auxiliary Function for generating individual weights for each unit-specific data entry. The method allows of opimtizing weights for all placebo and treated data configurations (assuming there are placebo configurations to generate)

Usage

```
synth_weights(
  data,
  time_window = NULL,
  custom_variable_weights = NULL,
  include_fit = FALSE,
  optimization_method = c("Nelder-Mead", "BFGS"),
  genoud = FALSE,
  quadopt = "ipop",
  Margin.ipop = 5e-04,
  Sigf.ipop = 5,
  Bound.ipop = 10,
  verbose = verbose,
  ...
)
```

Arguments

`data`	nested data of type `synth_tbl` generated from `sythetic_control()`. See `synthetic_control()` documentation for more information. In addition, a matrix of predictors must be pre-specified using the `generate_predictor()` function. See documentation for more information on how to generate a predictor function.
`time_window`	the temporal window of the pre-intervention outcome time series to be used in the optimization task. Default behavior uses the entire pre-intervention time period.
`custom_variable_weights`	a vector of provided weights that define a variable's importance in the optimization task. The weights are intended to reflect the users prior regarding the relative significance of each variable. Vector must sum to one. Note that the method is significantly faster when a custom variable weights are provided. Default behavior assumes no wieghts are provided and thus must be learned from the data.
`include_fit`	Boolean flag, if TRUE, then the optimization output is included in the outputted `tbl_df`.

`optimization_method`
: string vector that specifies the optimization algorithms to be used. Permissable values are all optimization algorithms that are currently implemented in the optimx function (see this function for details). This list currently includes c("Nelder-Mead', 'BFGS', 'CG', 'L-BFGS-B', 'nlm', 'nlminb', 'spg', and 'ucminf"). If multiple algorithms are specified, synth will run the optimization with all chosen algorithms and then return the result for the best performing method. Default is "BFGS". As an additional possibility, the user can also specify 'All' which means that synth will run the results over all algorithms in optimx.

`genoud`
: Logical flag. If true, synth embarks on a two step optimization. In the first step, genoud, an optimization function that combines evolutionary algorithm methods with a derivative-based (quasi-Newton) method to solve difficult optimization problems, is used to obtain a solution. See genoud for details. In the second step, the genoud results are passed to the optimization algorithm(s) chosen in optimxmethod for a local optimization within the neighborhood of the genoud solution. This two step optimization procedure will require much more computing time, but may yield lower loss in cases where the search space is highly irregular.

`quadopt`
: string vector that specifies the routine for quadratic optimization over w weights. possible values are "ipop" and "LowRankQP" (see ipop and LowRankQP for details). default is 'ipop'

`Margin.ipop`
: setting for ipop optimization routine: how close we get to the constraints (see ipop for details)

`Sigf.ipop`
: setting for ipop optimization routine: Precision (default: 7 significant figures (see ipop for details)

`Bound.ipop`
: setting for ipop optimization routine: Clipping bound for the variables (see ipop for details)

`verbose`
: Logical flag. If TRUE then intermediate results will be shown.

`...`
: Additional arguments to be passed to optimx and or genoud to adjust optimization.

Value

tibble data frame with optimized weights attached.

Index