Donor-Based Post-Shock Forecasting for Mean and Volatility Dynamics

Qiyang Wang

2026-10-07

Contents

1 Abstract **2**

2 Introduction **3**

2.1 Motivation . . . 3

2.2 Overview of the framework . . . 3

2.3 Contributions and relation to existing work . . . 4

2.4 Roadmap . . . 5

3 Methodology and Implementation **5**

3.1 Problem setup and notation . . . 5

3.2 Donor balancing weights (`dbw`) . . . 6

3.3 Synthetic mean forecasting (`SynthPrediction`) . . . 10

3.4 Structured donor pools (MultiPool extension) . . . 14

3.5 Adjustment for additional confounding shocks . . . 18

3.6 Volatility model selection (`auto_garchx`) . . . 22

3.7 Synthetic volatility forecasting (`SynthVolForecast`) . . . 24

4 Data and Empirical Setup **28**

4.1 Data Sources . . . 28

4.2 Distributed Package Objects . . . 28

4.3 Article Reproduction versus Data Reconstruction . . . 29

4.4 Construction of the Analysis Panel . . . 29

4.5 Inflation Adjustment and Reproducibility Limitations . . . 29

5 Simulation Study **30**

5.1 Targeted simulation for the postshock mean equation 30

5.2 Simulation for the volatility equation 33

5.3 Discussion 34

6 Empirical Application: ConocoPhillips under Large Shocks **35**

6.1 Target and donor shock episodes 36

6.2 MultiPool extension 38

6.3 Summary and discussion 39

7 Empirical Application: IYG Volatility around the 2016 U.S. Election **40**

7.1 Empirical setting and evaluation target 40

7.2 Candidate donor pools 40

7.3 Package workflow 41

7.4 Forecast comparison across donor-pool specifications 41

7.5 Discussion 42

8 Apple empirical application **43**

8.1 Standard package workflow with historical control 43

8.2 Modular and delayed-response sensitivity analyses 44

9 Conclusion **45**

10 References **46**

1 Abstract

Forecasts based only on pre-shock observations can perform poorly when a time series is hit by a large structural shock. This problem is especially relevant in financial applications, where abrupt events may affect both the conditional mean and the conditional variance of the target series. We propose a donor-based post-shock forecasting framework that uses historical shock episodes from related series to adjust forecasts after a new shock occurs.

The framework combines two forecasting modules with a common donor-weighting step. For conditional mean forecasting, donor-specific shock effects are estimated using ARIMA or ARIMAX models and then aggregated using donor balancing weights constructed from pre-shock covariates. For conditional volatility forecasting, the same idea is extended to GARCH-X models: donor-specific post-shock variance effects are estimated from historical shock episodes and transferred to the target's baseline pre-shock GARCH variance forecast. In both settings, the weighting step gives more influence to donor episodes that more closely resemble the target before the shock.

The framework also includes two extensions for more complex empirical settings. A multi-pool structure allows donors to be grouped into interpretable shock channels before aggregation, while a control-shock specification incorporates additional known events as structured regressors. Simulation studies show that the donor-adjusted forecasts can improve on unadjusted ARIMA and GARCH-type baselines when the target undergoes a post-shock shift. Empirical applications to ConocoPhillips during the March 2020 oil-market collapse, IYG volatility around the 2016 U.S. election, and an Apple control-shock example illustrate the use of the framework for mean forecasting, volatility forecasting, and control-shock analysis in financial data. The full workflow is implemented in the `postshock` R package and is designed to support reproducible empirical analysis.

2 Introduction

2.1 Motivation

Time series observed in economics, finance, environmental science, and related fields often experience large shocks. These shocks may come from policy announcements, supply disruptions, financial crises, geopolitical events, or sudden changes in demand. When such events occur, the pattern learned from pre-shock data may no longer describe the post-shock period well. A model fitted only to the pre-shock observations may therefore continue the old pattern into a new regime and produce biased forecasts.

This problem appears in both mean forecasting and volatility forecasting. In mean forecasting, a shock may cause a sudden level shift or a persistent change in the target trajectory. In volatility forecasting, the same shock may change the size, persistence, or clustering of future fluctuations. For example, a market event may not only move the expected return or price level of an asset, but also increase the uncertainty around future movements. A useful post-shock forecasting framework should therefore be able to handle both the conditional mean and the conditional variance.

In many empirical applications, the analyst often has access to more than the target series alone. Historical episodes from related series are often available, and some of these episodes may have experienced similar shocks in the past. These historical episodes can serve as donors. The basic idea is to estimate how donors responded after their own shocks and then use this information to adjust the forecast for the new target shock. The challenge is that donors are not equally useful. Some donors may look similar to the target before the shock, while others may differ substantially in their pre-shock behavior or covariate environment. A simple average of all donor effects can therefore be unstable or misleading.

2.2 Overview of the framework

This report develops a donor-based post-shock forecasting framework built around donor balancing weights. The framework first compares the target and donor episodes using information available before the shock. It then assigns larger weights to donors that are more similar to the target and uses the weighted donor effects to adjust the target forecast. In this way, the adjustment is not based on all donors equally, but on a synthetic donor combination designed to match the target more closely in the pre-shock period.

The donor-weighting step is implemented through `dbw()`. This component is model-agnostic: it constructs weights from pre-shock covariates and can, in principle, be combined with different baseline

forecasting models. In the current implementation, these weights are used in two main forecasting modules. The first module, `SynthPrediction()`, focuses on post-shock mean forecasting. It estimates donor-specific shock effects using ARIMA or ARIMAX models with post-shock indicators, combines those effects using the donor balancing weights, and adds the resulting adjustment to the target baseline forecast.

The second module, `SynthVolForecast()`, applies the same donor-adjustment idea to volatility forecasting. Instead of estimating post-shock changes in the conditional mean, this module estimates post-shock changes in the variance equation. For each donor episode, a GARCH-X model is fitted with a post-shock indicator, and the coefficient on this indicator is treated as the donor-specific volatility adjustment. The helper function `auto_garchx()` provides a systematic way to select GARCH-X specifications when the order is not supplied by the user. These donor-specific variance adjustments are then combined using the donor balancing weights and added to the target's baseline pre-shock GARCH-family variance forecast. This gives a parallel workflow for adjusting the target's post-shock variance forecast using historical donor episodes.

The framework also includes two extensions that are currently implemented mainly in the mean-forecasting workflow. First, the multi-pool extension allows donors to be grouped into interpretable categories before aggregation. For example, in an energy-market application, one may separate supply-related and demand-related donor shocks. Shock effects are first combined within each pool and then aggregated across pools. Second, the control-shock extension allows additional known events in the target window to be included as structured regressors. This is useful when the observed post-shock movement may reflect both the main shock of interest and other known events.

2.3 Contributions and relation to existing work

This report builds on the post-shock adjustment idea of Lin and Eck (2021). Their work studies the problem of forecasting a target series immediately after a structural shock by borrowing information from historical donor episodes whose post-shock outcomes are already observed. They propose several shock-effect aggregation estimators, including simple averaging, inverse-variance weighting, and similarity weighting, and analyze when these adjustments can reduce forecast risk. Their empirical examples focus on post-shock stock-price forecasting for ConocoPhillips and Apple in an autoregressive mean-forecasting setting.

The present work takes this idea as a starting point and develops it into a broader and reusable implementation. First, we implement donor balancing weights as a general weighting layer through `dbw()`, so that donor weights can be constructed from pre-shock covariates and reused across different forecasting modules. Second, we wrap the donor-adjustment workflow into `SynthPrediction()`, which supports ARIMA and ARIMAX baselines for post-shock mean forecasting. Third, we extend the same donor-adjustment logic to volatility forecasting through `SynthVolForecast()`, where donor-specific post-shock effects are estimated in a GARCH-X variance equation. Finally, we add multi-pool aggregation and control-shock specifications to handle more structured empirical settings.

The resulting `postshock` package is therefore not intended to replace the original post-shock adjustment idea, but to operationalize and extend it. The ConocoPhillips case study serves as a benchmark linked to the original application, while the IYG volatility and Apple control-shock examples illustrate additional directions made possible by the package implementation.

2.4 Roadmap

The remainder of the report is organized as follows. Section 3 introduces the problem setup, notation, and methodological components of the framework. Section 4 describes the data objects and empirical setup used in the applications. Section 5 presents simulation studies for both mean and volatility forecasting. Section 6 presents empirical applications of the mean-forecasting module using ConocoPhillips, including the baseline single-pool specification and the MultiPool extension. Section 7 presents a real-data volatility forecasting application using `SynthVolForecast()` for IYG around the 2016 U.S. election. Section 8 gives an Apple control-shock extension. Section 9 concludes.

3 Methodology and Implementation

3.1 Problem setup and notation

The primary objective of the `postshock` package is to address the failure of traditional forecasting models in the presence of **substantial structural shocks**. Under such "big shocks," baseline models (like ARIMA or GARCH) estimated on pre-shock data typically suffer from severe bias as the underlying data-generating process undergoes a radical shift. Our framework leverages a **donor-based counterfactual approach** to adjust these forecasts by borrowing "shock signatures" from historical donor episodes.

3.1.1 Regime Boundaries and Timing

We consider a target time series $Y_{0,t}$ and J donor series $Y_{j,t}$. We model the shock not as a point-event, but as a **regime shift**—a transition from a stable baseline to a post-shock state.

To facilitate the donor-based adjustment, all series are aligned relative to their respective shock inception points. For each series $j \in \{0, \dots, J\}$, we define T_0 as the **terminal boundary of the pre-shock regime**.

- **Pre-shock Regime**: The period $t \in \{1, \dots, T_0\}$ where the series follows its undisturbed dynamics.
- **Post-shock Regime**: The period $t > T_0$ where the "big shock" effect begins to materialize.

The index T_0 represents the **last "clean" observation** for the series in its truncated or aligned form. Thus, $T_0 + 1$ is the **inception of the shock effect**, regardless of whether the physical catalyst occurred exactly at a time index or in the interval between observations (e.g., overnight market news).

3.1.2 The Post-Shock Forecasting Objective

The framework is designed to correct the target's trajectory $Y_{0,T_0+1}, \dots, Y_{0,T_0+k}$ following a major disruption. We decompose the final forecast into a "no-shock" baseline and a donor-informed synthetic adjustment $\widehat{\omega}_h$:

$$\widehat{Y}_{0,T_0+h} = \underbrace{\widehat{Y}^{\text{baseline}}_{0,T_0+h}}_{\text{Standard ARIMA/GARCH}} + \underbrace{\widehat{\omega}_h}_{\text{Synthetic Shock Adjustment}}, \quad h = 1, \ldots, k$$

This decomposition explicitly targets the bias introduced by big shocks, ensuring that $\widehat{\omega}_h$ captures the magnitude of the regime shift that standard models would otherwise ignore.

3.1.3 Software Mapping

In the `postshock` package, the theoretical boundaries and regime definitions are operationalized through specific user-supplied arguments. To ensure the framework correctly captures the magnitude of the shock, the data must be aligned as follows:

- **Regime Boundaries (T_0):** Specified via the `shock_time_vec` argument. Each element j in this vector must be the **integer index of the last clean observation** for that specific series Y_j.
  - *Example:* `shock_time_vec <- c(500, 450, 600)`
- **Shock Observation Depth (L_j):** Specified via `shock_length_vec`. This defines the number of observations *after* T_0 (i.e., from T_0+1 to T_0+L_j) used to extract the "shock signature" from each donor.
  - *Example:* `shock_length_vec <- c(10, 10, 10)`
- **Data Structures:** The series and their respective pre-shock characteristics are passed as lists (`Y_series_list` and `covariates_series_list`). The order of elements in these lists must strictly match the order in `shock_time_vec`.

3.2 Donor balancing weights (`dbw`)

The first component of the framework is a donor balancing weights procedure, implemented in the function `dbw()`. The goal is to construct a weight vector

$$\omega = (\omega_1, \ldots, \omega_J)^\top$$

on the donor series such that a weighted combination of donors closely matches the target on a pre-shock window. Intuitively, donors that look more similar to the target before the shock should receive larger weights, while dissimilar donors should receive smaller (or zero) weights.

For exposition, suppose that each series is represented by a feature vector $z_i \in \mathbb{R}^K$ summarizing its pre-shock history, where $i = 0$ denotes the target and $i = 1, \ldots, J$ denote the donors. These features can include raw covariates at the last pre-shock time, multiple lookback values, and transformed auxiliary information, as described below. Let

$$z_0 \in \mathbb{R}^K \quad \text{and} \quad Z_0 = \begin{pmatrix} z_1^\top \\ \vdots \\ z_J^\top \end{pmatrix} \in \mathbb{R}^{J \times K}$$

collect the target feature vector and the donor feature matrix. A basic formulation of the donor balancing problem is then

$$\min_{\omega} \left\| z_0 - Z_0^\top \omega \right\|_p + \lambda \cdot \text{pen}(\omega) \quad \text{subject to} \quad \sum_{j=1}^{J} \omega_j = 1, \qquad L \le \omega_j \le U \text{ for all } j,$$

where $\|\cdot\|_p$ is either an ℓ_1 or ℓ_2 norm, $\lambda \ge 0$ is a regularization parameter, $\text{pen}(\omega)$ is an ℓ_1 (LASSO-type) or ℓ_2 (ridge-type) penalty on the weights, and L, U are lower and upper bounds. The simplex constraint $\sum_j \omega_j = 1$ together with nonnegative bounds ensures that ω can be interpreted as convex combination weights on the donors.

For clarity, we summarize the main notation used in this section. The integer J denotes the number of donor series, and K is the dimension of the feature vectors used for balancing. For each series i, the vector $z_i \in \mathbb{R}^K$ collects its pre-shock features, with z_0 corresponding to the target and $z_1, \dots, z_J$ to the donors. The matrix $Z_0 \in \mathbb{R}^{J \times K}$ stacks the donor features row-wise, so that $Z_0^\top \omega$ is the weighted donor feature vector under weights $\omega = (\omega_1, \dots, \omega_J)^\top \in \mathbb{R}^J$. The parameter $p \in \{1, 2\}$ determines whether an ℓ_1 or ℓ_2 norm is used in the matching loss, and $\lambda \ge 0$ controls the strength of the penalty $\text{pen}(\omega)$, which can be either ℓ_1-type (LASSO) or ℓ_2-type (ridge) depending on the argument `penalty_normchoice`. The constants L and U are lower and upper bounds on each weight, typically $L = 0$ and $U = 1$, while the equality constraint $\sum_{j=1}^{J} \omega_j = 1$ (corresponding to `sum_to_1 = TRUE`) enforces that ω defines a convex combination of the donor series.

In the `postshock` implementation, the input `X` is a list of data frames or matrices. The first element `X[[1L]]` contains the target series, and the remaining elements `X[[2L]]`, …, `X[[J+1L]]` contain the donors. Rows correspond to time and columns to features. The integer vector `dbw_indices` selects which feature columns are used in the matching objective. The vector `shock_time_vec` has the same length as `X` and records, for each series, the last pre-shock row; each series is truncated to rows `1:shock_time_vec[i]` before any further processing.

Several options control how the feature vectors z_i are constructed:

- The optional list `Y` can contain auxiliary series for each element of `X`, for example returns or transformed variables.
- The argument `Y_lookback_indices` specifies which backwards-looking rows of `Y[[i]]` are to be used (for example, `c(1, 2, 5)` corresponds to the last, second last, and fifth last pre-shock rows), and the function `inputted_transformation` is applied to each `Y[[i]]` before these lookback values are extracted.
- Similarly, `X_lookback_indices` can be used to extract multiple lookback rows from the truncated `X[[i]]`; if `X_lookback_indices` is `NULL`, only the final pre-shock row is used.

The selected rows are vectorized and concatenated to form a single feature vector for each series. Constant columns are removed automatically, and optional centering and scaling are applied via the logical arguments `center` and `scale`. If `princ_comp_count` is not `NULL`, a principal component analysis is performed and only the specified number of leading components are retained, which can help mitigate collinearity in high-dimensional settings.

Three arguments control distance and regularization. The argument `normchoice` is either `"l1"` or `"l2"`, corresponding to ℓ_1 or ℓ_2 distance in the matching loss. The argument `penalty_normchoice`

selects an ℓ_1 (`"l1"`) or ℓ_2 (`"l2"`) penalty on the weights, and `penalty_lambda` is constrained to be nonnegative and specifies the penalty strength. When `sum_to_1` is `TRUE` and the bounds `bounded_below_by` and `bounded_above_by` enforce nonnegative weights, the ℓ_1 norm of ω is (up to numerical tolerance) fixed at one; in this regime an ℓ_1 penalty only adds a constant to the objective and does not change the minimizer, so an ℓ_2 penalty is typically more effective under simplex constraints.

Formally, the optimization problem is solved using the `solnp` routine from the `Rsolnp` package, subject to the box constraints `bounded_below_by` and `bounded_above_by` and the optional equality constraint `sum_to_1`. The function returns a list with three main components: `opt_params`, the estimated donor weight vector $\hat{\omega}$; `loss`, the final matching loss; and `convergence`, a flag indicating whether the numerical optimizer reports successful convergence. These weights are then used inside `SynthPrediction()` and `SynthVolForecast()` to combine donor-specific forecasts for the target series.

In the illustrative examples and empirical applications below, we often use a convex-combination specification with ℓ_2-regularized weights. A typical configuration is:

- `sum_to_1 = TRUE`
- `bounded_below_by = 0`
- `bounded_above_by = 1`
- `normchoice = "l2"`
- `penalty_normchoice = "l2"`
- `penalty_lambda = 1e-2`

Under this configuration, the weights can be interpreted as nonnegative donor shares that sum to one and therefore define a convex combination. The ℓ_2 penalty stabilizes the solution when donors are highly correlated. This specification corresponds to a constrained ridge-type matching problem on the simplex.

The following example illustrates the computation of donor weights using three pre-shock features.

```
set.seed(1)

T  <- 50
K  <- 3

# Construct a target and two donors with K=3 features
target <- data.frame(
  x1 = rnorm(T),
  x2 = rnorm(T,  0.2),
  x3 = rnorm(T, -0.1)
)
donor1 <- data.frame(
  x1 = rnorm(T,  0.1),
  x2 = rnorm(T, -0.1),
  x3 = rnorm(T,  0.3)
)
donor2 <- data.frame(
```

```
  x1 = rnorm(T, -0.2),
  x2 = rnorm(T,  0.4),
  x3 = rnorm(T,  0.0)
)

X_list <- list(target, donor1, donor2)
shock  <- c(40, 40, 40)   # last pre-shock row for each series

dbw_out <- dbw(
  X                 = X_list,
  dbw_indices       = 1:K,
  shock_time_vec    = shock,
  center            = TRUE,
  scale             = TRUE,
  sum_to_1          = TRUE,
  bounded_below_by  = 0,
  bounded_above_by  = 1,
  normchoice        = "l2",
  penalty_normchoice = "l2",
  penalty_lambda    = 1e-2
)

cat("Estimated donor weights:\n")
```

```
## Estimated donor weights:
```

```
print(round(dbw_out$opt_params, 3))
```

```
## [1] 0.17 0.83
```

```
cat("\nFinal matching loss:\n")
```

```
##
## Final matching loss:
```

```
print(dbw_out$loss)
```

```
## [1] 2.804706
```

```
cat("\nConvergence flag:\n")
```

```
##
## Convergence flag:
```

```r
print(dbw_out$convergence)
```

```
## [1] "convergence"
```

3.3 Synthetic mean forecasting (`SynthPrediction`)

The second component of the framework is a synthetic mean–forecasting procedure, implemented in the function `SynthPrediction()`. The aim is to produce k-step-ahead forecasts for a target series by combining information from a pool of donor series that have experienced structural shocks in the past. The procedure operates in four layers: (i) estimation of donor-specific shock effects via (S)ARIMA models with a post-shock indicator; (ii) construction of donor weights (either equal or DBW-based); (iii) fitting an ARIMA model to the pre-shock portion of the target; and (iv) adjusting the target forecast by the weighted donor shock effect.

For notation, let $Y_{i,t}$ denote the value of series $i = 0, 1, \dots, J$ at time t, where $i = 0$ is the target and $i = 1, \dots, J$ are donors. For each series we observe a shock start time T_i and a post-shock window length L_i. In the implementation these are supplied via the vectors `shock_time_vec` and `shock_length_vec`, and are first converted to integer indices that match the time index of `Y_series_list`.

3.3.1 Donor-level (S)ARIMA with post-shock indicators

For each donor $i = 1, \dots, J$, we model the series as a baseline time-series component plus a structural shock effect. Specifically, we consider the decomposition

$$Y_{i,t} = \mu_{i,t} + \gamma_i D_{i,t} + \varepsilon_{i,t},$$

where $\mu_{i,t}$ follows an ARIMA or seasonal ARIMA process, $D_{i,t}$ is a post-shock indicator defined by

$$D_{i,t} = \begin{cases} 0, & t \le T_i, \\ 1, & T_i < t \le T_i + L_i, \end{cases}$$

and $\varepsilon_{i,t}$ denotes the innovation term. The parameter γ_i captures the average level shift induced by the shock over the specified window.

In the implementation, `SynthPrediction()` estimates this model using `forecast::auto.arima()` (or a user-specified `arima_order`), with a design matrix whose columns consist of the post-shock indicator and any selected covariates. The function `decide_seasonal()` determines whether a seasonal component should be included based on the detected or user-specified period. For automatic ARIMA, this period is carried in the response's `ts` frequency; the engine selects the orders, and need not retain a seasonal term. An explicit `seasonal_period` is a finite positive integer measured in observation steps. Period one, failed detection, or `seasonal = FALSE` disables seasonal search. `seasonal_order` applies only to a fixed `arima_order`. The target period is determined from its pre-shock fitting window, never its evaluation response. A donor automatic-fit error retains the explicitly non-seasonal ARIMA(1,1,1) fallback. When donor covariates enter the response model, selected X and Y must be complete, finite and aligned in the original fitting window. Invalid rows or indexed-date mismatches stop before either fitting or fallback: no automatic deletion, imputation or time-grid compression is performed.

If the argument `covariate_indices` is non-`NULL`, the donor design matrix contains both the post-shock indicator and the selected columns of `covariates_series_list[[i]]` as contemporaneous regressors. If `covariate_indices` is `NULL`, the only regressor is the post-shock indicator. After fitting the model, the donor shock effect $\hat{\gamma}_i$ is extracted as the coefficient associated with the post-shock indicator regressor (i.e., the post-shock indicator) using `lmtest::coeftest()`. Collecting these into a vector

$$\hat{\gamma} = (\hat{\gamma}_1, \dots, \hat{\gamma}_J)^\top,$$

we obtain donor-level estimates of how each past shock affected its corresponding series over the specified window.

3.3.2 Donor weighting

In the second layer, `SynthPrediction()` constructs donor weights $w = (w_1, \dots, w_J)^\top$. If `use_dbw = FALSE`, equal weights $w_j = 1/J$ are used. If `use_dbw = TRUE`, the function calls `dbw()` to compute data-driven weights based on pre-shock covariates.

For this purpose, the object `X_for_dbw` is formed by taking `covariates_series_list` (if available) or falling back to `Y_series_list`. Each element is converted to a numeric matrix. If `dbw_indices` is `NULL`, informative columns are selected automatically by examining cross-series variance at the last pre-shock row and discarding nearly constant columns. The resulting indices, together with the integer shock times, are passed to `dbw()` along with options such as `dbw_scale`, `dbw_center`, `penalty_lambda`, and `penalty_normchoice`. When `dbw()` succeeds, the optimal weights $\widehat{w}$ are taken from `dbw_output$opt_params`. With `princ_comp_input = NULL`, the existing warning/equal-weight fallback on a thrown DBW error or unavailable features is retained. An explicit PCA request instead stops on preprocessing or DBW errors, unknown/failed string status, or unusable weights. PCA counts must be finite positive integers and cannot exceed the actual SVD direction count after preprocessing; the limit is not a numerical-rank cutoff. `days_before_shocktime_vec` is one common lookback length; `geometric_sets` remains a reserved, ignored parameter.

The combined donor shock effect used for adjustment is then

$$\hat{\theta} = \sum_{j=1}^{J} \widehat{w}_j \hat{\gamma}_j,$$

which aggregates individual donor effects according to their DBW weights.

3.3.3 Target (S)ARIMA and shock adjustment

In the third layer, we fit an ARIMA model to the target series using only pre-shock data. Let T_0 denote the last pre-shock time for the target. `SynthPrediction()` restricts the target series to observations $t = 1, \dots, T_0$ and fits an ARIMA or seasonal ARIMA model in the same manner as for the donors, using `auto.arima()` or a user-specified `arima_order`. If `covariate_indices` is non-`NULL`, the selected columns of `covariates_series_list[[1]]` are lagged by one period (via `lag.xts`) and included as `xreg` for model fitting. For forecasting, the last pre-shock covariate row is replicated k times to form the `newxreg` matrix, so that the k-step-ahead forecasts correspond to holding the covariates fixed at their last observed values.

Let $\widehat{Y}^{\text{raw}}_{0,T_0+h}$ denote the resulting raw forecast for the target at horizon $h = 1, \dots, k$. The adjusted "DBW version" forecast is defined by

$$\widehat{Y}^{\text{adj}}_{0,T_0+h} = \widehat{Y}^{\text{raw}}_{0,T_0+h} + \hat{\theta},$$

which adds the weighted donor shock effect $\hat{\theta}$ to the raw ARIMA forecast. This adjustment relies on the identifying assumption that the weighted donor shock effect provides a consistent proxy for the counterfactual shock effect on the target under comparable structural conditions. For comparison, the function also computes an "arithmetic mean" version that uses the simple average of donor effects,

$$\widehat{Y}^{\text{arith}}_{0,T_0+h} = \widehat{Y}^{\text{raw}}_{0,T_0+h} + \frac{1}{J}\sum_{j=1}^{J} \hat{\gamma}_j.$$

The outputs of `SynthPrediction()` are collected in a list with three main components. The element `linear_combinations` contains the donor weight vector $\widehat{w}$; `predictions` contains three forecast vectors labelled `unadjusted`, `adjusted`, and `arithmetic_mean`; and `meta` stores additional information such as the donor-specific $\hat{\gamma}_j$, the combined effect $\hat{\theta}$, the ARIMA orders, and the information criterion used. These objects are used in Section 4 to construct the empirical results.

3.3.3.1 Illustrative simulation example To illustrate how `SynthPrediction()` behaves in a simple setting, we simulate one target series and two donor series observed on a daily grid with $T = 200$. The target Y_0 is a random walk around level 100. Donor 1 is a very close copy of the target (small extra noise), while donor 2 is much noisier. As covariates we construct a second random walk ("base signal") and create two highly informative covariates X_0, X_1 plus one largely uninformative covariate X_2. We treat time index $T_0 = 150$ as the shock time for all series and estimate a one–step–ahead forecast ($k = 1$).

```r
set.seed(123)

## 1) Target and donors --------------------------------
T     <- 200
dates <- seq.Date(as.Date("2021-01-01"), by = "day", length.out = T)

# target: random walk around level 100
y0 <- cumsum(rnorm(T, mean = 0, sd = 1)) + 100
Y0 <- xts(y0, order.by = dates)

# donor 1: very similar to target (small extra noise)
y1 <- y0 + rnorm(T, mean = 0, sd = 0.1)
Y1 <- xts(y1, order.by = dates)

# donor 2: noisier copy of target
y2 <- y0 + rnorm(T, mean = 0, sd = 2)
Y2 <- xts(y2, order.by = dates)

Y_list <- list(Y0, Y1, Y2)
```

```r
## 2) Covariates ---------------------------------------
base_signal <- cumsum(rnorm(T, mean = 0, sd = 1))   # common latent factor

X0 <- xts(base_signal + rnorm(T, sd = 0.1), order.by = dates)
X1 <- xts(base_signal + rnorm(T, sd = 0.1), order.by = dates)
X2 <- xts(cumsum(rnorm(T, sd = 2)),         order.by = dates)  # less informative

X_list <- list(X0, X1, X2)

## 3) Shock specification -----------------------------
shock_time_vec   <- c(150, 150, 150)  # last pre-shock boundary is row 150
shock_length_vec <- c(1L,  1L,  1L)   # one-step post-shock window

## 4) Run SynthPrediction -----------------------------
res <- SynthPrediction(
  Y_series_list          = Y_list,
  covariates_series_list = X_list,
  shock_time_vec         = shock_time_vec,
  shock_length_vec       = shock_length_vec,
  k                      = 1,
  covariate_indices      = 1,
  seasonal               = FALSE
)

res$linear_combinations      # DBW donor weights
```

```
## [1] 0.9880241 0.0119759
```

```r
res$predictions              # raw / adjusted forecasts
```

```
## $unadjusted
## [1] 96.35265
##
## $adjusted
## [1] 97.2558
##
## $arithmetic_mean
## [1] 97.461
```

```r
# True value of Y0 at T0+1 (time 151)
Y_list[[1]][150:151]
```

```
##                [,1]
## 2021-05-30 96.34557
## 2021-05-31 97.13331
```

```r
truth <- as.numeric(Y_list[[1]][151])

c(
  truth      = truth,
  raw        = res$predictions$unadjusted[1],
  dbw_adj    = res$predictions$adjusted[1],
  arith_mean = res$predictions$arithmetic_mean[1]
)
```

```
##      truth        raw    dbw_adj arith_mean
##   97.13331   96.35265   97.25580   97.46100
```

3.4 Structured donor pools (MultiPool extension)

In some applications, the donor set may contain shocks generated by different economic mechanisms. For example, the ConocoPhillips application contains both financial-crisis-related shocks and oil-supply-related shocks. Treating all donors as a single homogeneous pool may hide this structure and make the final adjustment harder to interpret.

The MultiPool extension is implemented as a structured wrapper around the existing SynthPrediction() workflow. It does not define a separate forecasting model. Instead, when the argument pools is provided, SynthPrediction() dispatches to the MultiPool routine, which repeatedly applies the standard SynthPrediction() workflow to user-specified donor subsets. Each subset represents an interpretable shock channel, such as a demand-side or supply-side donor group. The resulting pool-level shock adjustments are then combined using a user-specified aggregation rule, such as summation or weighted averaging.

This design keeps the baseline forecasting machinery unchanged. The single-pool SynthPrediction() workflow remains the core computational unit, while MultiPool adds an additional aggregation layer on top of it. In this sense, MultiPool should be interpreted as a structured donor-pool organization mechanism rather than a separate model class.

3.4.1 Pool structure and notation

Let $Y_{i,t}$, $i = 0, 1, \dots, J$, denote the target series ($i = 0$) and donor series ($i = 1, \dots, J$) observed over time t. Donors are grouped into G **potentially overlapping pools**

$$\mathcal{J}_1, \dots, \mathcal{J}_G,$$

supplied as a named list

$$\texttt{pools} = \{\mathcal{J}_g\}_{g=1}^{G},$$

where each element contains donor series names that match `names(Y_series_list)`. The first element of `Y_series_list` is always treated as the target.

> **Practical requirement.** Since pools refer to donors by name, `Y_series_list` (and, if provided, `covariates_series_list`) must be named with consistent ordering.

3.4.2 Step 1: Common baseline forecast (unique counterfactual)

The procedure first computes a **single baseline forecast** by calling `SynthPrediction()` on the full donor set (without pool structure). This defines the unique counterfactual forecasting path used throughout the multi-pool procedure and produces a raw k-step-ahead forecast

$$\widehat{Y}^{\text{raw}}_{0,T_0+h}, \quad h = 1, \dots, k,$$

stored as `predictions$unadjusted`.

Whether donor balancing weights (DBW) are used in this baseline run is controlled by `base_use_dbw`.

> **Role of Step 1.** Step 1 defines the only baseline counterfactual forecast. All multi-pool variants differ only in how the shock adjustment term is constructed, not in how the raw counterfactual path is generated.

3.4.3 Step 2: Pool-specific shock adjustments (channel extraction)

For each pool g, the algorithm:

1. Subsets the series list to include the target and the donors in $\mathcal{J}_g$.
2. Runs `SynthPrediction()` on this restricted subset, optionally using DBW within the pool (controlled by `pool_use_dbw`).
3. Extracts the pool-level combined donor shock quantity

   $$\hat{\theta}_g \equiv \hat{\omega}_g,$$

   obtained from the subset output's `meta$combined_omega`.

Each $\hat{\theta}_g$ is interpreted as a **pool-specific shock adjustment** capturing the contribution of the structural channel represented by pool g under comparable pre-shock conditions. Importantly, the pool-specific runs are **not used as alternative forecasts**; they serve only as devices to extract pool-level shock adjustments under donor restrictions.

Collecting across pools yields

$$\hat{\theta} = (\hat{\theta}_1, \dots, \hat{\theta}_G)^\top.$$

> **DBW at baseline vs within pools.** DBW at baseline enforces global pre-shock balance using the full donor set. DBW within pools enforces balance conditional on the pool restriction. These serve distinct roles and should not be interpreted as "double balancing" the same objective.

3.4.4 Step 3: Cross-pool aggregation (constructing the adjustment)

The pool-level shock adjustments $\hat{\theta}$ are aggregated into a single scalar adjustment term $\hat{\theta}$ using the pool_agg rule.

- **Sum aggregation**

$$\hat{\theta} = \sum_{g=1}^{G} \hat{\theta}_g.$$

- **Weighted aggregation** Given user-specified weights π_g,

$$\hat{\theta} = \sum_{g=1}^{G} \pi_g \hat{\theta}_g.$$

  The weighted case requires a named numeric vector aligned with pool names.

- **Modeling choice.** Sum aggregation corresponds to an additive structural decomposition of the target shock effect across channels. Weighted aggregation imposes a user-specified contribution profile. Neither aggregation rule is implied by the data; both encode modeling/identification assumptions.

3.4.5 Step 4: Multi-pool adjusted forecast

The final multi-pool forecast is defined as

$$\widehat{Y}^{\text{adj,mp}}_{0,T_0+h} = \widehat{Y}^{\text{raw}}_{0,T_0+h} + \hat{\theta}, \quad h = 1, \dots, k.$$

This adjusted series is stored as `predictions$adjusted_multipool`.

Importantly, the baseline raw forecast is unchanged; only the shock adjustment term differs across pooling strategies.

3.4.6 Output structure

The multi-pool extension preserves the original output of `SynthPrediction()` and adds:

- `predictions$adjusted_multipool`,
- `meta$multi_pool`, which contains
  - the pool definitions,
  - pool-specific shock adjustments $\{\hat{\theta}_g\}$,
  - aggregation rule and (optional) weights,
  - aggregated adjustment $\hat{\theta}$,
  - indicators of whether DBW was used at baseline or within pools.

This design enables **decomposition analysis**, allowing practitioners to attribute portions of the final adjustment to distinct structural channels as encoded by the donor pools.

3.4.7 Identification interpretation

The multi-pool framework generalizes the single-pool identifying assumption by allowing donor heterogeneity to be represented explicitly through pool structure.

Within each pool g, pooled donor shock adjustment $\hat{\theta}_g$ is assumed to provide a valid proxy for the corresponding structural component of the target's counterfactual shock effect under comparable pre-shock conditions.

When sum aggregation is used, an additional **additive separability** assumption is invoked:

$$\theta_{\text{target}} = \sum_{g=1}^{G} \theta_g.$$

Under this assumption, the aggregated adjustment $\hat{\theta}$ provides a reduced-form estimate of the total shock adjustment constructed from the sum of pool-level components. This hierarchical structure relaxes the homogeneous-donor assumption of the baseline method while preserving interpretability and computational tractability.

```r
## MultiPool example: reuse the same simulated data above -----------------

# 0) IMPORTANT: MultiPool requires named lists (pools refer to donors by name)
names(Y_list) <- c("target", "donor_close", "donor_noisy")
names(X_list) <- names(Y_list)

# 1) Define donor pools (channels)
pools <- list(
  close_copy = c("donor_close"),
  noisy_copy = c("donor_noisy")
)

# 2) Example pool weights (must align with pool names)
pool_weights <- c(
  close_copy = 0.5,
  noisy_copy = 0.5
)

# 3) Run MultiPool with weighted aggregation
res_mp <- SynthPrediction(
  Y_series_list          = Y_list,
  covariates_series_list = X_list,
  shock_time_vec         = shock_time_vec,
  shock_length_vec       = shock_length_vec,
  k                      = 1,
  covariate_indices      = 1,
  seasonal               = FALSE,
  pools                  = pools,
  pool_agg               = "weighted",
  pool_weights           = pool_weights,
```

```
  base_use_dbw           = TRUE,
  pool_use_dbw           = TRUE
)

# Inspect pool effects + aggregated effect
res_mp$meta$multi_pool$pool_effects
```

```
## close_copy noisy_copy
##  0.8981187  1.3185822
```

```
res_mp$meta$multi_pool$aggregated_effect
```

```
## [1] 1.10835
```

```
# Compare forecasts
c(
  truth          = truth,
  raw            = res_mp$predictions$unadjusted[1],
  single_adj_dbw = res_mp$predictions$adjusted[1],
  multipool_adj  = res_mp$predictions$adjusted_multipool[1]
)
```

```
##          truth            raw single_adj_dbw  multipool_adj
##       97.13331       96.35265       97.25580       97.46100
```

3.5 Adjustment for additional confounding shocks

In empirical applications, the primary shock of interest may not be the only event affecting the target series during the post-shock period. Additional confounding shocks—such as subsequent product releases, policy announcements, or macroeconomic disturbances—may occur within or shortly after the main shock window. If left unaccounted for, these confounding shocks can bias the estimated shock adjustment and distort counterfactual forecasts.

To address this issue, the proposed framework allows **explicit control for additional confounding shocks** through structured shock regressors included in the forecasting model.

3.5.1 Confounding shock specification

Let T_0 denote the onset of the primary shock of interest. Suppose that additional confounding shocks occur at times

$$T_1, T_2, \ldots, T_M,$$

with corresponding durations

$$L_1, L_2, \ldots, L_M.$$

For each confounding shock $m = 1, \dots, M$, we construct a deterministic shock indicator

$$D_{m,t} = \mathbf{1}\{T_m \leq t < T_m + L_m\},$$

or, more generally, a user-specified shock profile (e.g., step, pulse, or finite window effect). These indicators are collected into a matrix of confounding shock regressors

$$\mathbf{D}_t = (D_{1,t}, \dots, D_{M,t})^\top.$$

The last pre-shock boundary and effect length are supplied through `shock_time_vec` and `shock_length_vec`. Additional controls are target-only: `control_shocks[[1]]` is passed to `build_control_xreg()`, for example `list(list(list(time = 3, length = 1, shape = "point")))`. All forecast-time control values are zero. Their inclusion does not establish causal removal of a confounding event.

3.5.2 Forecasting model with confounding shocks

Let $Y_{0,t}$ denote the target series. Conditional on the donor-based counterfactual structure, the forecasting equation is augmented to include confounding shocks as exogenous regressors:

$$Y_{0,t} = \mu_t + \alpha\, \mathbf{1}\{t \geq T_0\} + \gamma^\top \mathbf{D}_t + \varepsilon_t,$$

where

- μ_t represents the baseline mean component;
- α captures the primary shock effect;
- γ captures the effects of additional control shocks;
- ε_t is an idiosyncratic error term.

The control-shock regressors are included in the target-side forecasting model so that known additional events can be accounted for when fitting the baseline dynamics. This reduces the risk that the donor-based adjustment is interpreted as capturing movements that are actually associated with other known events.

3.5.3 Relation to additive shock decomposition

This approach is closely related to empirical strategies that decompose observed outcomes into additive effects of multiple structural shocks. For example, existing work in macroeconomics and energy economics studies supply- and demand-driven shocks by modeling their effects as additive components of the observed response.

In contrast to structural VAR approaches—where additivity parameters are estimated within a fully parametric system—the present framework adopts a reduced-form perspective. Confounding shocks are controlled via explicit exogenous regressors, while donor-based counterfactuals capture the underlying baseline dynamics. Additivity across shocks is imposed as an identifying assumption rather than inferred from a structural model.

3.5.4 Interaction with MultiPool adjustment

The control-shock adjustment is conceptually separate from the MultiPool extension described in the previous section. Specifically:

1. MultiPool decomposes the primary shock effect across heterogeneous donor channels.
2. Confounding shock controls help account for subsequent or overlapping events.

When both mechanisms are used, the final adjusted forecast takes the form

$$\widehat{Y}_{0,T_0+h} = \widehat{Y}^{\text{raw}}_{0,T_0+h} + \hat{\theta} + \gamma^\top \mathbf{D}_{T_0+h},$$

where $\hat{\theta}$ is the aggregated MultiPool adjustment.

This layered structure allows practitioners to disentangle (i) baseline counterfactual dynamics, (ii) primary shock effects decomposed across donor channels, and (iii) additional confounding shocks, within a unified forecasting pipeline.

3.5.5 Practical considerations

- Confounding shocks should be declared only when their timing and qualitative interpretation are well understood.
- Over-specification of shock controls may reduce efficiency, particularly in short post-shock samples.
- When multiple confounding shocks overlap in time, estimated coefficients should be interpreted jointly rather than individually.

Overall, explicit adjustment for additional confounding shocks enhances the credibility and interpretability of post-shock forecasts, particularly in settings where multiple discrete events affect the target series within a narrow time window.

```r
set.seed(123)

## 1) Target and donors ---------------------------------
T     <- 200
dates <- seq.Date(as.Date("2021-01-01"), by = "day", length.out = T)

# target: random walk around level 100
y0 <- cumsum(rnorm(T, mean = 0, sd = 1)) + 100

# donor 1: very similar to target (small extra noise)
y1 <- y0 + rnorm(T, mean = 0, sd = 0.1)

# donor 2: noisier copy of target
y2 <- y0 + rnorm(T, mean = 0, sd = 2)

Y_list <- list(
```

```
  target = xts(y0, order.by = dates),
  donor1 = xts(y1, order.by = dates),
  donor2 = xts(y2, order.by = dates)
)

## 2) Covariates --------------------------------------
base_signal <- cumsum(rnorm(T, mean = 0, sd = 1))

X_list <- list(
  target = xts(base_signal + rnorm(T, sd = 0.1), order.by = dates),
  donor1 = xts(base_signal + rnorm(T, sd = 0.1), order.by = dates),
  donor2 = xts(cumsum(rnorm(T, sd = 2)),         order.by = dates)
)

## 3) Shock specification -----------------------------
t_main <- 150L
shock_time_vec   <- rep(t_main, length(Y_list))
shock_length_vec <- rep(1L,     length(Y_list))

## 4) Run SynthPrediction (no confounding control) -----
res <- SynthPrediction(
  Y_series_list          = Y_list,
  covariates_series_list = X_list,
  shock_time_vec         = shock_time_vec,
  shock_length_vec       = shock_length_vec,
  k                      = 1,
  covariate_indices      = 1,
  seasonal               = FALSE
)

# 1-step truth (time t_main+1)
truth <- as.numeric(Y_list$target[t_main + 1])

c(
  truth      = truth,
  raw        = as.numeric(res$predictions$unadjusted[1]),
  adjusted   = as.numeric(res$predictions$adjusted[1]),
  arith_mean = as.numeric(res$predictions$arithmetic_mean[1])
)
```

```
##      truth        raw   adjusted arith_mean
##   97.13331   96.35265   97.25580   97.46100
```

```
## 5) OPTIONAL: add one confounding shock on the target only ----
# Example: a short confounding event before the main shock
t_conf   <- 120L
len_conf <- 5L
```

```r
control_shocks <- vector("list", length(Y_list))
control_shocks[[1]] <- list(list(
  time = t_conf, length = len_conf, shape = "window"
))

res_ctl <- SynthPrediction(
  Y_series_list          = Y_list,
  covariates_series_list = X_list,
  shock_time_vec         = shock_time_vec,
  shock_length_vec       = shock_length_vec,
  k                      = 1,
  covariate_indices      = 1,
  control_shocks         = control_shocks,
  seasonal               = FALSE
)

c(
  truth        = truth,
  raw          = as.numeric(res_ctl$predictions$unadjusted[1]),
  adjusted_ctl = as.numeric(res_ctl$predictions$adjusted[1])
)
```

```
##        truth          raw adjusted_ctl
##     97.13331     96.23859     97.14174
```

3.6 Volatility model selection (`auto_garchx`)

The next component of the framework is a volatility–forecasting module based on GARCH–type models with exogenous regressors. It is implemented in the function `auto_garchx()`, which selects a GARCHX specification by BIC and optionally refits the selected model using a more stable backcast for the conditional variance.

Let $\{y_t\}_{t=1}^{n}$ denote a (demeaned) return series and let x_t collect exogenous regressors (if any). Following the parametrisation of the `garchx` package, we consider models of the form

$$y_t = \sigma_t \varepsilon_t, \qquad \varepsilon_t \sim \text{i.i.d. } (0,1),$$

with conditional variance

$$\sigma_t^2 = \omega + \sum_{i=1}^{p} \alpha_i y_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2 + \sum_{\ell=1}^{o} \gamma_\ell y_{t-\ell}^2 \mathbf{1}\{y_{t-\ell} < 0\} + \delta^\top x_t.$$

Here q is the GARCH order, p is the ARCH order, and the coefficients γ_ℓ capture GJR–type asymmetry. In the `garchx` implementation these orders are supplied as an integer vector `order` $= (q, p, o)$, where $o \in \{0, 1\}$ indicates whether the asymmetric GJR term is included ($o = 1$) or not ($o = 0$).

The main arguments of `auto_garchx()` are:

- `y`: numeric vector of returns or residuals (no `NA`/`Inf`/`NaN`);
- `xreg`: optional matrix of exogenous regressors aligned with `y`;
- `max.p`, `max.q`: upper bounds for the ARCH (`p`) and GARCH (`q`) orders;
- `search_o`: subset of $\{0, 1\}$ indicating whether symmetric (`o = 0`) and/or asymmetric (`o = 1`) specifications are considered;
- `final_refit`: logical flag controlling whether a second "backcast refit" step is applied to the selected model;
- `clamp_factor`: two–element numeric vector controlling the range used to clamp the unconditional variance relative to the sample variance;
- `vcov.type`: type of standard errors requested from `garchx` (`"ordinary"` or `"robust"`).

3.6.1 Grid search over (q, p, o) by BIC

The function `auto_garchx()` performs a grid search over the order triple (q, p, o). Given user–supplied maxima `max.q`, `max.p` and the set `search_o` $\subseteq \{0, 1\}$, the function loops over

$$q = 0, \ldots, \texttt{max.q}, \quad p = 0, \ldots, \texttt{max.p}, \quad o \in \texttt{search_o},$$

and fits a `garchx` model for each candidate specification. A candidate participates only when it returns without error, its top-level `convergence` is a finite numeric scalar equal to zero, its coefficients are a nonempty finite real vector, and BIC is finite. Missing or unknown flags are rejected; finite standard errors are not a prerequisite. Strictly lower BIC replaces the incumbent, so exact ties retain the first eligible candidate. If none qualifies, selection stops with categorized rejection counts. When `final_refit = TRUE`, the existing unconditional-variance backcast calculation and skip conditions are retained. A same-order refit replaces the qualified grid winner only if it passes these same checks and strictly improves BIC; otherwise the original grid fit is retained.

3.6.2 Illustrative simulation example

```r
## Simulate a GARCH(1,1) series
set.seed(123)
n      <- 2000
omega  <- 0.05
alpha  <- 0.07
beta   <- 0.90

eps    <- rnorm(n)
sigma2 <- numeric(n)
y      <- numeric(n)

sigma2[1] <- omega / (1 - alpha - beta)
y[1]      <- sqrt(sigma2[1]) * eps[1]

for (t in 2:n) {
  sigma2[t] <- omega + alpha * y[t-1]^2 + beta * sigma2[t-1]
  y[t]      <- sqrt(sigma2[t]) * eps[t]
```

```
}

## Run auto_garchx
best <- auto_garchx(
  y,
  max.p       = 3,
  max.q       = 3,
  search_o    = c(0L, 1L),
  final_refit = TRUE,
  vcov.type   = "robust",
  verbose     = FALSE
)

## Show selected orders and BIC
c(p = best$p, q = best$q, o = best$o)
```

```
## p q o
## 1 1 0
```

```
best$bic
```

```
## [1] 6571.432
```

```
coef(best$fit)
```

```
##  intercept      arch1     garch1
## 0.04597810 0.06060302 0.91151343
```

3.7 Synthetic volatility forecasting (`SynthVolForecast`)

The last component of the framework is a *synthetic volatility–forecasting* module, implemented in the function `SynthVolForecast()`.
The goal is to produce k-step–ahead **variance** forecasts for a target series by combining a standard pre-shock GARCH model with donor-based post-shock variance adjustments.

For one target series and J donors we observe demeaned returns $\{y_{i,t}\}_{t=1}^{n}$ and, for donors, a set of contemporaneous covariates $x_{i,t}$. For each series we also observe a shock time T_i and a post–shock window length L_i, supplied via the vectors `shock_time_vec` and `shock_length_vec`, and collected into the list `Y_series_list` (target in position 1, donors in positions 2,…,$J+1$).

3.7.1 Donor–level GARCH–X with post–shock indicators

For each donor $i = 1, \dots, J$ we fit a GARCH–type model with a 0/1 post–shock indicator. Following the parametrisation of the `garchx` package, we consider

$$y_{i,t} = \sigma_{i,t}\varepsilon_{i,t}, \qquad \varepsilon_{i,t} \sim \text{i.i.d. } (0,1),$$

with conditional variance

$$\sigma_{i,t}^2 = \omega_i + \sum_{\ell=1}^{p_i} \alpha_{i,\ell} y_{i,t-\ell}^2 + \sum_{\ell=1}^{q_i} \beta_{i,\ell} \sigma_{i,t-\ell}^2 + \sum_{\ell=1}^{o_i} \gamma_{i,\ell} y_{i,t-\ell}^2 \mathbf{1}\{y_{i,t-\ell} < 0\} + \delta_i^\top x_{i,t} + \eta_i D_{i,t},$$

where q_i, p_i, and o_i are the GARCH, ARCH, and asymmetry orders, respectively. Automatic selection searches $o_i \in \{0, 1\}$. The post–shock indicator $D_{i,t}$ is defined by

$$D_{i,t} = \begin{cases} 0, & t \le T_i, \\ 1, & T_i < t \le T_i + L_i, \\ 0, & t > T_i + L_i. \end{cases}$$

In the implementation we build the design matrix for donor i by stacking the selected donor covariates (columns chosen via `covariate_indices`) together with `post_shock_indicator`, whose coefficient corresponds to η_i. The function then fits either

- a user–supplied GARCH order `garch_order = c(q,p,o)`, or
- if `garch_order = NULL`, a data–driven specification chosen by `auto_garchx()` via BIC over a grid in (q, p, o).

After fitting, the donor–specific post–shock effect $\widehat{\omega}_i^* := \widehat{\eta}_i$ is extracted using `lmtest::coeftest()` if available, and stored in the vector

$$\widehat{\omega}^* = (\widehat{\omega}_1^*, \dots, \widehat{\omega}_J^*)^\top.$$

3.7.2 Donor weighting via DBW

To transfer information from donors to the target we form a covariate collection

$$X_{\text{for dbw}} = (X_{\text{target}}, X_1, \dots, X_J),$$

where X_{target} is the covariate matrix for the target (`target_covariates`) and X_j is the donor–j covariate matrix (`covariates_series_list[[j]]`). The function `dbw()` is then used to compute donor weights $w = (w_1, \dots, w_J)^\top$ by solving a regularised balancing problem on the pre–shock covariates. The main tuning parameters are

- `dbw_indices` : which covariate columns enter the balancing problem;
- `penalty_lambda` and `penalty_normchoice` : regularisation weight and norm;
- `dbw_scale` and `dbw_center` : whether to standardise covariates.
- `princ_comp_input`: optional number of leading principal components passed to `dbw()`; if `NULL`, PCA is not used.

The resulting combined donor shock effect is

$$\widehat{\theta} = \sum_{j=1}^{J} w_j \widehat{\omega}_j^*,$$

which aggregates donor–level post–shock effects according to their DBW weights.

3.7.3 Target GARCH and variance adjustment

For the target series we fit a **pre-shock** GARCH model *without* exogenous regressors. Let T_0 denote the last pre-shock time for the target. The function `SynthVolForecast()` restricts the target data to $t = 1, \dots, T_0$ and fits either a user-specified `garch_order` or a BIC-selected specification via `auto_garchx()`. Because no `xreg` is used for the target, forecasting does not require a future `newxreg` matrix.

Let $\hat{v}^{\text{raw}}_{0,T_0+h}$ denote the resulting k-step-ahead **variance** forecast at horizon $h = 1, \dots, k$. In the implementation, the DBW-adjusted volatility forecast is defined as

$$\hat{v}^{\text{adj}}_{0,T_0+h} = \max\left\{\hat{v}^{\text{raw}}_{0,T_0+h} + \hat{\theta}, \epsilon\right\}, \qquad \epsilon = 10^{-8}.$$

This adjustment adds the combined donor post-shock effect $\hat{\theta}$ to the raw GARCH variance forecast while enforcing a strictly positive variance forecast. For comparison, the function also computes an arithmetic-mean adjustment that ignores DBW and uses the simple average of donor effects,

$$\hat{v}^{\text{arith}}_{0,T_0+h} = \max\left\{\hat{v}^{\text{raw}}_{0,T_0+h} + \frac{1}{J}\sum_{j=1}^{J}\hat{\omega}^*_j, \epsilon\right\}.$$

The outputs of `SynthVolForecast()` are collected in a list with three main components. The element `linear_combinations` contains the donor weight vector $\hat{w}$; `predictions` contains three forecast vectors labelled `unadjusted`, `adjusted`, and `arithmetic_mean`; and `meta` stores additional information such as the donor-specific $\hat{\omega}^*_j$, the combined effect $\hat{\theta}$, the selected GARCH orders, the donor-specific standard errors, and the DBW convergence status. When `return_fits = TRUE`, the fitted donor and target GARCH objects are also returned for diagnostic inspection.

Forecast accuracy is evaluated outside `SynthVolForecast()` using realized variance proxies, such as next-period squared returns. This separation keeps the forecasting function focused on variance prediction and avoids treating the latent conditional variance as directly observed.

3.7.4 Illustrative simulation example

```r
set.seed(123)

## 1) Simulate one target and two donor return series ----------------------
n    <- 500
eps0 <- rnorm(n)
eps1 <- rnorm(n)
eps2 <- rnorm(n)

# simple homoskedastic DGP for illustration
y0 <- eps0                      # target
y1 <- 0.8 * eps0 + 0.2 * eps1   # donor 1, highly correlated with target
y2 <- 0.5 * eps0 + 0.5 * eps2   # donor 2, less correlated

Y_list <- list(y0, y1, y2)
```

```r
## 2) Simple covariates for target and donors ----------------------------

X1 <- cbind(x1 = rnorm(n), x2 = rnorm(n))  # donor 1 covariates
X2 <- cbind(x1 = rnorm(n), x2 = rnorm(n))  # donor 2 covariates
Xt <- cbind(x1 = rnorm(n), x2 = rnorm(n))  # target covariates

## 3) Shock times and window lengths -------------------------------------
shock_time_vec   <- c(350L, 350L, 350L)
shock_length_vec <- c(20L,  20L,  20L)

## 4) Call SynthVolForecast ----------------------------------------------
out_vol <- SynthVolForecast(
  Y_series_list          = Y_list,
  covariates_series_list = list(X1, X2),
  target_covariates      = Xt,
  shock_time_vec         = shock_time_vec,
  shock_length_vec       = shock_length_vec,
  k                      = 5,
  covariate_indices      = 1:2,
  dbw_indices            = 1:2,
  return_fits            = FALSE
)

## 5) Show number of donors, weights, and forecasts ----------------------
out_vol$meta$n_donors
```

```
## [1] 2
```

```r
out_vol$linear_combinations
```

```
## [1] 5.125626e-09 1.000000e+00
```

```r
out_vol$predictions
```

```
## $unadjusted
## [1] 0.8920638 0.9152028 0.9166576 0.9157751 0.9147486
##
## $adjusted
## [1] 0.9887246 1.0118637 1.0133185 1.0124360 1.0114094
##
## $arithmetic_mean
## [1] 0.9864278 1.0095668 1.0110216 1.0101392 1.0091126
```

4 Data and Empirical Setup

4.1 Data Sources

The empirical workflows retain the four original frozen processed objects and add a derived `aapl_s1_input` in version 0.2.1. The Apple and IYG acquisition notebooks use Yahoo Finance through `quantmod::getSymbols()`; CPIAUCSL is obtained from FRED's CSV endpoint. CPIAUCSL is produced by the U.S. Bureau of Labor Statistics and distributed through the Federal Reserve Bank of St. Louis (https://fred.stlouisfed.org/series/CPIAUCSL). The accompanying article cites the quantmod software and the CPI series.

The Apple notebook uses closing prices for AAPL and the S&P 500 (^GSPC), and the closing ^IRX rate. It requests data from 1 January 2000, with no fixed end date. The IYG notebook uses log returns of adjusted IYG and CL=F prices, the ^VIX close, and changes in ^IRX and ^TNX closes. Its requested interval runs from the earliest episode date minus 2,000 calendar days to the latest episode date plus 60 days. Neither notebook records the exact original retrieval timestamp or a frozen raw-provider snapshot.

For COP, package documentation records Yahoo/quantmod market data and FRED CPI. Its processed fields represent COP, S&P 500, WTI, the dollar index, a Treasury bill rate, and VIX. A complete original COP object-writing script was not located in the supplied project. In particular, the WTI instrument identifier and the Treasury-bill provider-series mapping cannot be independently verified from the numeric object. A different research notebook uses FRED DTB3; this does not establish that COP's `TB_Close` uses DTB3 rather than the ^IRX series previously described in this report. These provenance limits do not alter the locked COP inputs or reported workflow.

4.2 Distributed Package Objects

- `cop_oil_postshock` contains six 60-row target/donor windows, five matching features, and local boundary/effect-length vectors.
- `iyg_onestep_input` contains the target return series and the `restricted`, `elections_only`, and `all` donor-pool inputs.
- `aapl_onestep_input` contains dated target/donor panels, `Z_tgt_base`, and control/effect-date metadata used to prepare the approved standard S1 inputs.
- `aapl_control_input` contains an additional dated Apple panel used for provenance and overlap checks, not a separate current main benchmark.
- `aapl_s1_input` contains the verified S1 fitting/matching inputs, explicit settings, dated boundaries, and a separate realized evaluation observation. It is derived only from the two older Apple objects, which remain unchanged.

Each object is loaded with `data(..., package = "postshock")`. Date metadata must be interpreted explicitly rather than assuming all stored shock indices share the forecasting API's boundary convention.

4.3 Article Reproduction versus Data Reconstruction

The primary reproduction layer starts from these frozen package objects and runs the current package functions with the recorded dependencies. It requires no market-data download. Forecasts, tables, and errors are generated from executable results, with rounding only at display time. The shared Apple logic is installed as `replication/apple_diagnostic.R`. It carries the article supplement's shared analysis and unchanged numerical assertions, with an S1 input adapter for `aapl_s1_input`. This report locates it with `system.file()`. The installed implementation manifest checks function bodies as well as the package version to identify the implementation used for reproduction. The derived-input builder is `data-raw/aapl_s1_input.R` in the source working tree; its implementation is also installed as `replication/apple_data_builder.R`. No adjacent manuscript directory is needed.

The second layer traces how raw sources were acquired and transformed. It is a provenance/reconstruction pathway, not a guarantee that a future live download will recreate the object bit for bit. The supplementary *Data provenance and reconstruction* document records source evidence, variable conventions, construction steps, missing acquisition records, and a safe isolated-reconstruction procedure. Whole research notebooks should not be sourced indiscriminately: they contain exploratory calculations and package-data overwrite commands.

4.4 Construction of the Analysis Panel

The Apple and IYG notebooks inner-join their selected/transformed series by trading date and remove incomplete rows. Index alignment remains an analyst responsibility; forecasting functions cannot repair a semantically incorrect effect date or lagged-control alignment.

COP uses boundary row 30 and evaluation row 31, associated with 6 and 9 March 2020. Donor mean models use the first 31 observations. IYG construction takes 500 observations before the episode date and ten after, placing the date at row 501. The variance evaluation proxy is the following squared return, row 502; historical donor variance fits use the full supplied series with a one-observation effect indicator. The stored COP and IYG numeric objects do not retain complete independent calendar-date indices.

For Apple S1, `Z_tgt_base` supplies 25 observations from 7 October through 10 November 2020. Donor effect dates are 7 June 2017, 31 October 2018, and 4 June 2019. Their local effect row is 13; the API boundary is therefore row 12. The target boundary is row 25, and all effect lengths are one. S1 uses ten-observation GSPC/IRX histories through each boundary for matching. The stored `aapl_onestep_input$shock_time_vec` instead points to the original target evaluation/donor effect observations; it is not passed unchanged to the standard workflow. The shared script derives the approved boundaries without modifying the object.

4.5 Inflation Adjustment and Reproducibility Limitations

The inspected Apple code computes calendar-year means of monthly CPI and uses

$$a_y = \frac{\overline{\mathrm{CPI}}_{2020}}{\overline{\mathrm{CPI}}_y}$$

to scale AAPL and GSPC observations for year y. IRX is not scaled. Non-finite annual factors are dropped, but the notebook does not enforce twelve observations per retained CPI year. The IYG return pipeline does not apply this annual scaling.

COP documentation describes localized 2020-base inflation adjustment. Without its complete builder, the exact scaled-column set is unverified; the earlier blanket statement that all prices and covariates become 2020-dollar quantities is not supported. The frozen data are retained, not reprocessed to impose a new convention.

Annual-average CPI and later historical downloads may contain information unavailable at the original forecast origin. Consequently these are retrospective illustrations, not strict real-time-vintage backtests. Vendor revisions, absent retrieval timestamps/raw snapshots, and missing COP/older-Apple object writers prevent a promised exact live reconstruction. Reproduction of the reported experiments instead uses frozen objects, recorded software versions, and explicit numerical checks.

5 Simulation Study

In this section we provide a targeted simulation study to illustrate how the proposed postshock forecasting framework behaves in controlled settings. The purpose of these simulations is not to deliver a comprehensive benchmark over all possible designs, but rather to verify that the adjustment mechanisms behave as expected under stylized postshock data-generating processes. We consider one set of simulations for the postshock mean equation and one for the postshock volatility equation.

All simulations below run from the code embedded in this report. The companion exploratory notebooks `sim-test.Rmd` and `sim_vol_tets.Rmd` are retained in the same source folder for background, but are not sourced and are not required to reproduce the report. No old chunk cache is used.

5.1 Targeted simulation for the postshock mean equation

5.1.1 Design

For the mean equation, we use a stylized postshock simulation setup designed to mirror the empirical forecasting problem while remaining simple and interpretable. In each simulation, we generate one target series and two donor series over 200 time points, with a known shock occurring at time index 150. We focus on one-step-ahead postshock forecasting.

The donor series are constructed to differ in their similarity to the target. One donor is generated by adding only small Gaussian noise to the target path and is therefore closely aligned with the target in the pre-shock period. The second donor is generated with substantially larger noise and is intentionally less aligned. This allows us to assess whether the weighting mechanism places greater emphasis on the more informative donor.

The mean simulations implemented in `sim-test.Rmd` consider two complementary design blocks. In the first block, we study covariate informativeness by constructing one informative covariate together with additional noisy covariates. The informative covariate is generated from a common latent signal with small perturbations, whereas the noisy covariates are generated independently as random-walk-type processes. This design is used to examine how forecast performance changes as the covariate set becomes less informative.

In the second block, we study several representative postshock patterns: a one-period local jump, a persistent level shift, and a volatility-type disturbance added after the shock date. These scenarios are designed to check whether the proposed adjustment procedure responds in the expected direction under simple but interpretable departures from the pre-shock trajectory.

Throughout, the shock date and local window structure are treated as known.

5.1.2 Forecasting procedures and evaluation criteria

For each simulated design, we construct the target and donor series together with their aligned covariates and pass them to `SynthPrediction()` using the same basic workflow as in the empirical application. We focus on one-step-ahead postshock prediction ($k = 1$).

We compare three forecasts for the target series:

1. **Unadjusted forecast.**
   The baseline ARIMA/ARIMAX model is fitted using only the pre-shock segment of the target series, and the one-step-ahead postshock prediction is obtained without any donor-based shock adjustment.

2. **Arithmetic-mean adjusted forecast.**
   The baseline prediction is augmented by the simple average of the estimated donor shock effects.

3. **DBW-adjusted forecast.**
   The baseline prediction is augmented by a donor-balancing-weights (DBW) combination of the estimated donor shock effects, where the donor weights are chosen to better match the target's pre-shock covariate environment.

Performance is assessed by comparing the predicted postshock value with the realized postshock value of the target series. In the covariate-sensitivity experiments, we summarize forecast errors and predicted-versus-realized values across alternative covariate designs. In the shock-pattern experiments, we compare the direction and magnitude of the forecast adjustments under different stylized postshock scenarios.

5.1.3 Reproducibility

The following executable chunk contains the reported mean-equation simulation. The separate `sim-test.Rmd` is an exploratory companion, not a runtime dependency.

Table 1: Absolute forecast error across alternative covariate designs.

k	Method	Error
1	ARIMA	0.529
1	Arithmetic mean	0.626
1	DBW adjusted	0.441

k	Method	Error
3	ARIMA	0.383
3	Arithmetic mean	0.364
3	DBW adjusted	0.175
5	ARIMA	0.496
5	Arithmetic mean	0.438
5	DBW adjusted	0.275

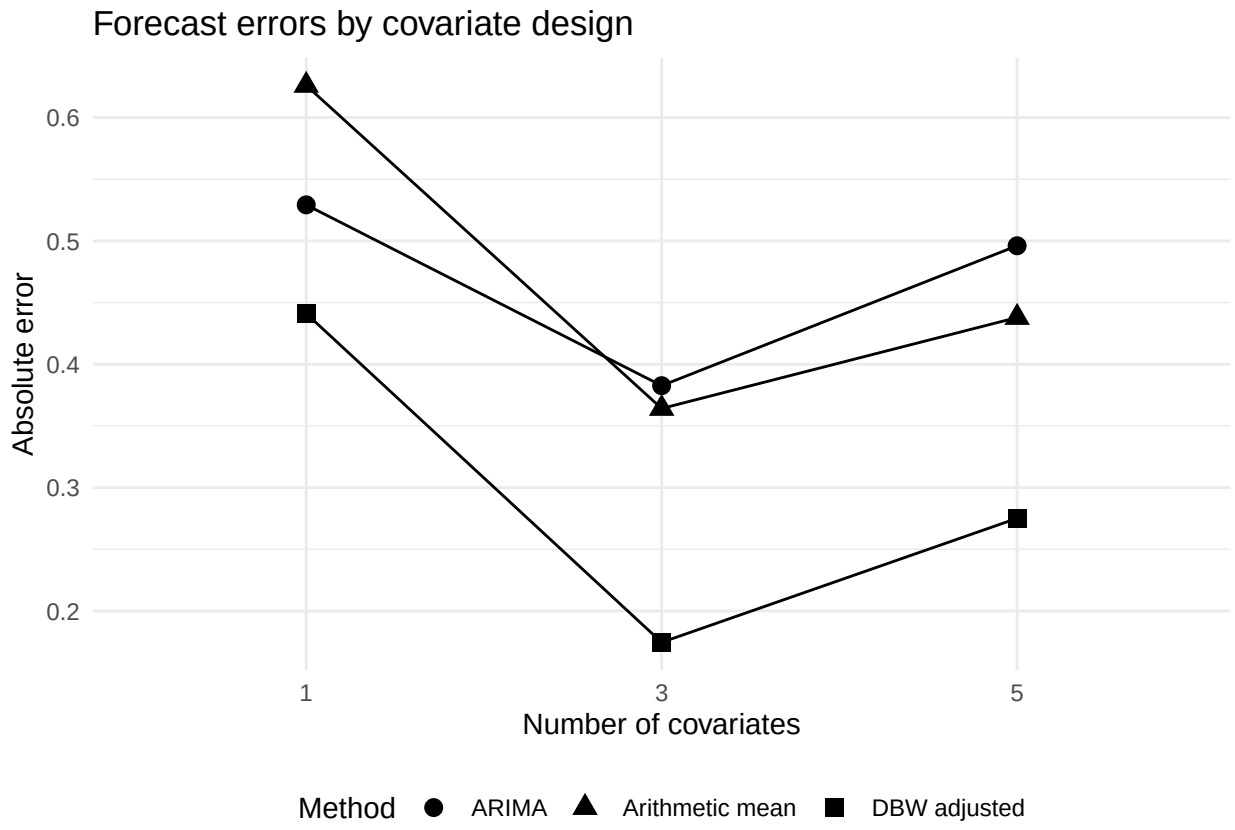
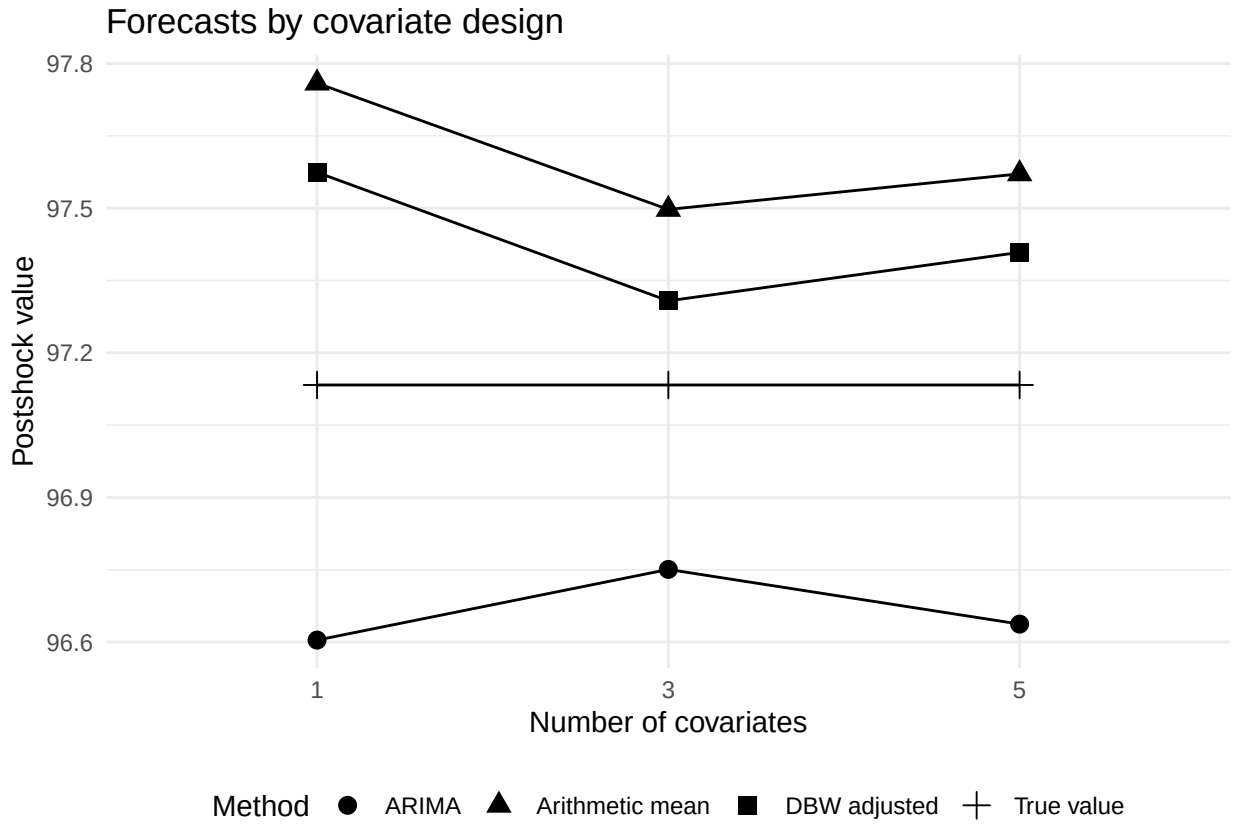


Table 2: True and predicted postshock values across covariate designs.

k	Method	Value
1	True value	97.133
1	ARIMA	96.604
1	Arithmetic mean	97.759
1	DBW adjusted	97.574
3	True value	97.133
3	ARIMA	96.751
3	Arithmetic mean	97.497
3	DBW adjusted	97.308

k	Method	Value
5	True value	97.133
5	ARIMA	96.637
5	Arithmetic mean	97.571
5	DBW adjusted	97.408



5.2 Simulation for the volatility equation

5.2.1 Estimators and evaluation criteria

For each simulated data set, we first use `auto_garchx()` to select a GARCHX-type specification for each series by BIC, allowing for both symmetric and asymmetric (GJR-type) variance dynamics. The selected model may then be refitted using the backcast-based stabilization step implemented in `auto_garchx()`. The target model is fitted through its boundary; historical donor models use their supplied series with a shock-window regressor. Based on these fitted volatility models, we apply `SynthVolForecast()` to construct postshock variance forecasts for the target series.

In the volatility simulations reported here, we compare two forecasts for the target variance:

1. **Unadjusted volatility forecast.**
   A pre-shock GARCHX-type model selected by `auto_garchx()` is fitted to the target series,

and its postshock conditional variance forecast is extrapolated without donor-based correction.

2. **DBW-adjusted volatility forecast.**
   The unadjusted variance forecast is augmented by a donor-based postshock adjustment, where donor shock effects are combined using donor balancing weights constructed from pre-shock volatility features.

Performance is evaluated by comparing the predicted conditional variance with the true conditional variance generated by the simulation design. In the representative omega-jump design, we report step-by-step true variance, raw forecast, adjusted forecast, and their corresponding absolute forecast errors over the short postshock horizon.

5.2.2 Results

The representative volatility simulation under the omega-jump design shows that the unadjusted pre-shock GARCH-type forecast tends to underreact to the increase in conditional variance after the shock. In contrast, the DBW-adjusted forecast moves closer to the true postshock variance over the short forecast horizon.

Table 3 reports the step-by-step true conditional variance together with the raw and DBW-adjusted forecasts. The figure below visualizes the same comparison and shows that the DBW-adjusted path tracks the true variance more closely than the raw baseline forecast.

These results are consistent with the intended role of the synthetic volatility module: when donor episodes are weighted according to pre-shock similarity in volatility features, donor information can improve short-horizon postshock variance forecasting relative to a purely pre-shock GARCH-type extrapolation.

Table 3: Volatility simulation summary for the omega-jump design.

Step	True	Raw	DBW	AE, raw	AE, DBW
1	0.289223	0.235606	0.302921	0.053617	0.013698
2	0.313962	0.256670	0.323985	0.057292	0.010023

5.3 Discussion

The simulation studies for the postshock mean and volatility components provide controlled settings in which the underlying shock effects and conditional moments are known. In these settings, the unadjusted baseline forecasts tend to underreact to structural changes because they are estimated using only pre-shock information and therefore extrapolate the pre-shock dynamics too smoothly into the postshock period.

For the mean component, incorporating donor information can improve postshock prediction, but the quality of the adjustment depends on how well the donor episodes resemble the target in the pre-shock period. The targeted simulations show that simple unweighted adjustments can become

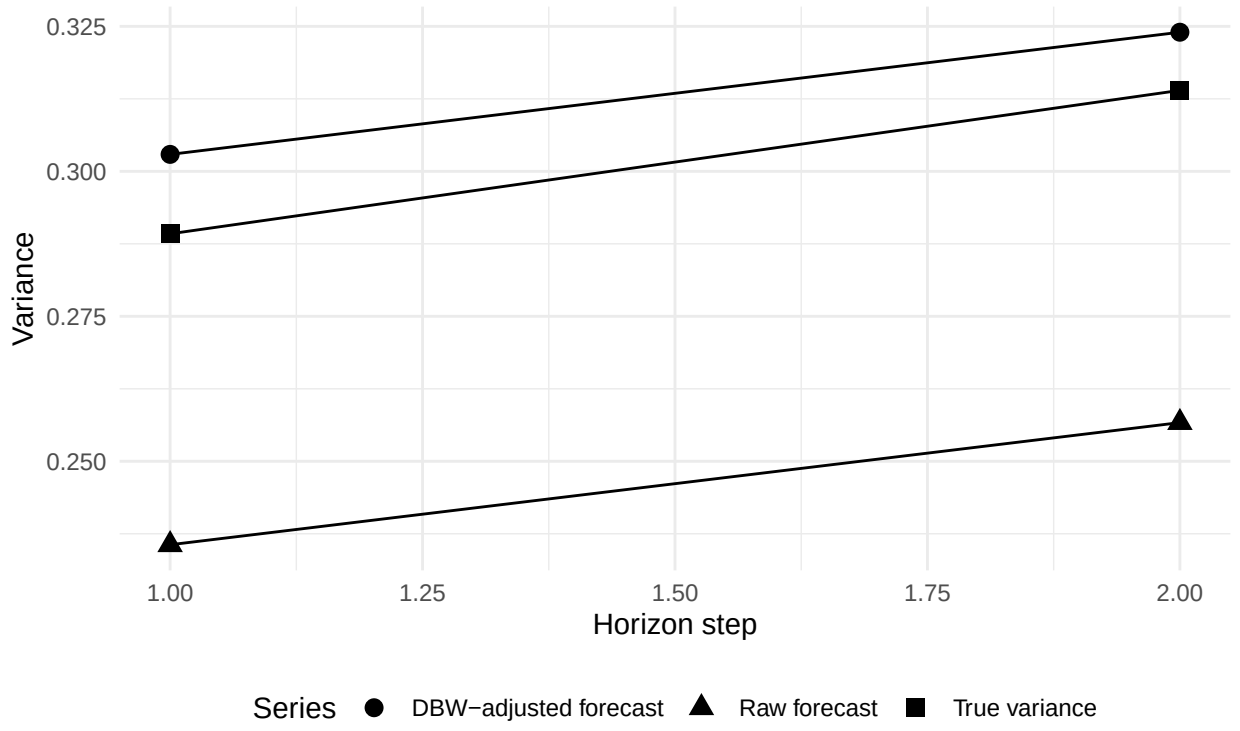


Figure 1: True and predicted variance under the omega-jump design.

less stable when donor series or covariates are noisy, whereas the DBW-based adjustment remains more robust by placing greater emphasis on donors that are better aligned with the target.

For the volatility component, the representative omega-jump designs show a similar pattern. The unadjusted pre-shock GARCH-type variance forecast tends to underpredict the postshock increase in conditional variance, while the DBW-adjusted forecast tracks the true postshock variance more closely over the short forecast horizon. This supports the use of pre-shock volatility features for donor weighting in the synthetic volatility module.

Taken together, these simulation results support the role of donor balancing weights as a general device for improving postshock forecasting in both the mean and volatility settings. They also provide a controlled complement to the empirical analysis in the next section, where the same workflow is applied to the ConocoPhillips case study.

6 Empirical Application: ConocoPhillips under Large Shocks

In this section we apply the postshock forecasting framework developed in Sections 2–3 to the ConocoPhillips (COP) data set described in Section 4. The goal is to forecast COP around large negative shocks by borrowing information from earlier donor shock episodes that occurred under different market conditions. We treat the COVID-19 oil price collapse in March 2020 as the target episode and use several earlier crisis episodes as donors.

6.1 Target and donor shock episodes

We focus on the large negative shock to ConocoPhillips on 9 March 2020 as the target episode. In the implementation we follow the convention of Section 3 and treat the last pre-shock trading day as the index T_0. For the March 2020 event this corresponds to 6 March 2020, so the shock itself occurs between T_0 and $T_0 + 1$.

As donor episodes we select five earlier crisis windows for COP. These correspond to the March 2008 Bear Stearns episode, three major September 2008 crisis episodes during the Lehman and bailout turmoil, and the late-November 2014 OPEC-related oil price collapse. For each episode we construct a local window of 30 trading days before T_0 and 30 trading days after the shock, and we work with inflation-adjusted prices expressed in 2020 U.S. dollars, as described in Section 4. ## Empirical implementation using the packaged data object

To keep the empirical workflow fully reproducible and consistent with the package interface, we use the processed object `cop_oil_postshock` distributed with the package rather than reconstructing the windows manually in the main text. This object contains the aligned target and donor shock windows, the corresponding macro-financial covariate matrices, and the shock timing metadata required by the forecasting functions.

In particular, `cop_oil_postshock` stores the outcome series in `Y`, the aligned covariate matrices in `X`, the covariate names in `covar_names`, and the local shock timing information in `shock_time_vec` and `shock_length_vec`. The empirical analysis therefore follows the same workflow as the general package implementation and can be reproduced directly from the processed object.

Using the same processed object, we next examine whether separating donor episodes into multiple pools provides a more interpretable or more accurate postshock adjustment for the March 2020 COP episode.

Table 4: Forecast comparison for the March 2020 COP shock episode.

Method	Value
Actual	34.070
Unadjusted	44.734
Arithmetic mean	39.750
DBW adjusted	39.111

Table 5: DBW donor weights for the COP empirical application.

Donor	Weight
Y_2008_03_17	0.000
Y_2014_11_28	0.637
Y_2008_09_09	0.000
Y_2008_09_15	0.000
Y_2008_09_29	0.363

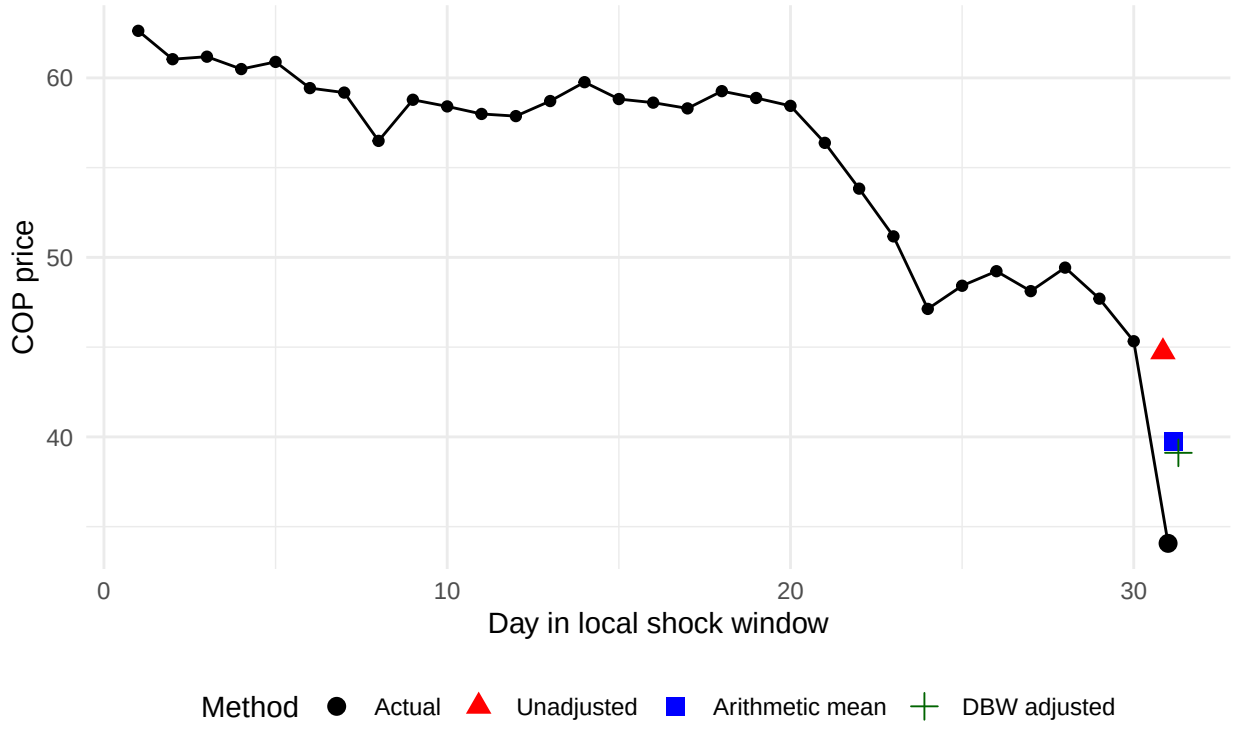


Figure 2: COP target path and one-step forecast comparison. Forecast markers refer to the same observation; horizontal displacement is for display only.

6.2 MultiPool extension

6.2.1 Motivation

The single-pool donor adjustment treats all historical donor episodes as belonging to a common donor set. In the COP application, however, this may be restrictive because the March 2020 collapse plausibly reflected multiple channels, including oil-market supply disruptions and broader demand or financial-stress effects. A MultiPool extension allows these channels to be represented separately by grouping donor episodes into economically interpretable pools and aggregating their contributions afterwards.

6.2.2 Pool definition

Using the same COP shock windows as in the baseline empirical analysis, we partition the donor episodes into two pools. The late-November 2014 episode is assigned to a **supply** pool because it is naturally associated with the OPEC-related oil-price collapse. The March and September 2008 episodes are assigned to a **demand** pool, as they are more closely associated with crisis-driven market stress and demand contraction.

The analyst-defined pool membership is:

- **Supply:** 28 November 2014.
- **Demand / financial stress:** 17 March, 9 September, 15 September, and 29 September 2008.

6.2.3 MultiPool implementation

We apply the MultiPool version of `SynthPrediction()` using the same aligned COP outcome and covariate series as in the baseline case. In this experiment, donor balancing weights are applied within each pool (`pool_use_dbw = TRUE`), while the baseline target forecast is kept as the unweighted ARIMA/ARIMAX component (`base_use_dbw = FALSE`).

We consider two aggregation rules across pools:

1. **Sum aggregation** (`pool_agg = "sum"`), which adds the estimated pool-level shock effects;
2. **Weighted aggregation** (`pool_agg = "weighted"`), which combines the pool-level shock effects using user-specified pool weights.

This comparison allows us to assess whether separating supply-side and demand-side donor pools yields a more interpretable or more accurate postshock adjustment than the single-pool specification.

Table 6: Forecast comparison for the COP empirical application under MultiPool specifications.

Method	Value
Actual	34.070
Unadjusted	44.734
Single-pool DBW	39.111
MultiPool (sum)	33.176
MultiPool (weighted)	34.331

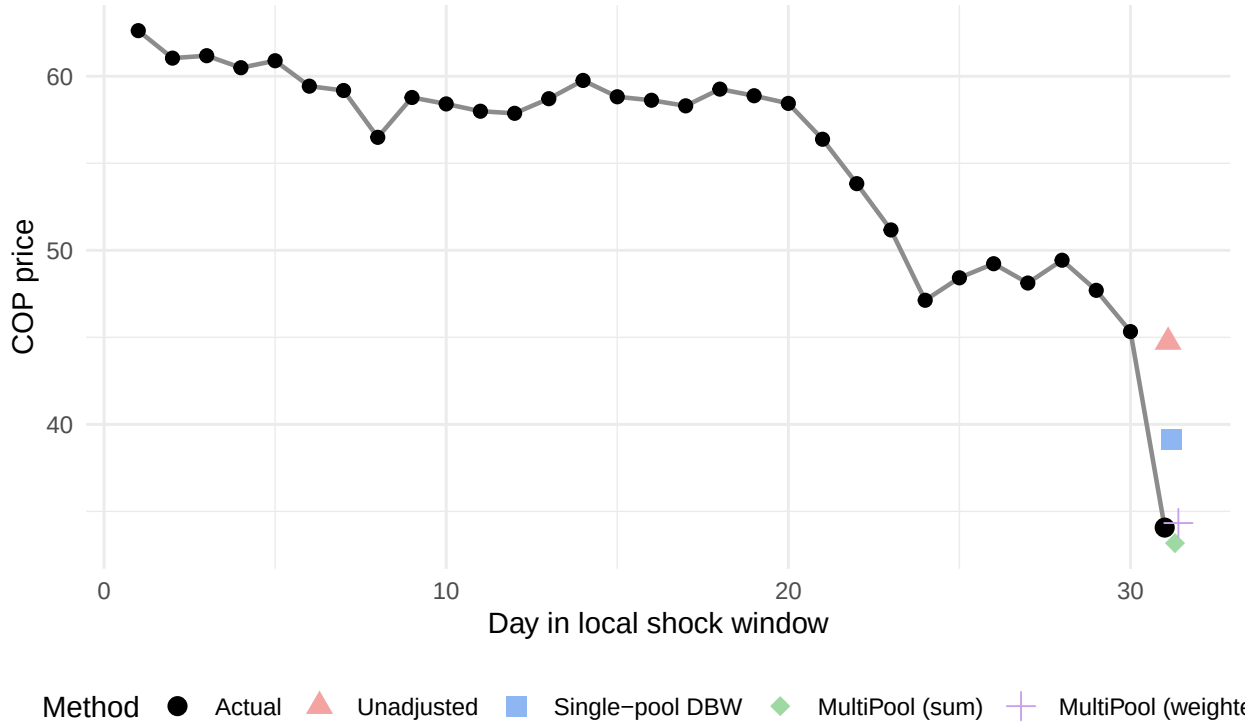


Figure 3: COP target path under single-pool and MultiPool specifications. Forecast markers refer to the same observation; horizontal displacement is for display only.

6.3 Summary and discussion

In this empirical section we applied the postshock forecasting framework to the ConocoPhillips data set using the processed package object `cop_oil_postshock`. The COVID-19 oil-price collapse of 9 March 2020 was treated as the target episode, while five earlier crisis windows from 2008 and 2014 served as donor episodes. The packaged object contains the aligned target and donor outcome series, the corresponding macro-financial covariate matrices, and the associated shock metadata, allowing the empirical workflow to be reproduced directly through the package interface.

The baseline empirical results illustrate the contribution of donor information relative to the unadjusted ARIMAX forecast. The unadjusted forecast, obtained by extrapolating the pre-shock target

dynamics without any donor-based correction, lies substantially above the realised COP price on the shock date. Both the arithmetic-mean adjustment and the DBW-adjusted forecast move closer to the realised value, with the DBW-adjusted version performing slightly better in this case. This supports the idea that reweighting donor episodes according to pre-shock similarity can improve postshock prediction relative to simple averaging.

The MultiPool extension further enriches the empirical analysis by allowing donor episodes to be partitioned into economically interpretable groups. In the COP application, the donor episodes were separated into supply-related and demand-related pools, reflecting the view that the March 2020 collapse likely combined multiple transmission channels. The MultiPool results show that this extension can provide a more structured and interpretable decomposition of the postshock adjustment, while also yielding forecasts that are competitive with, and in this case slightly closer to the realised value than, the single-pool DBW forecast.

Overall, the COP case study demonstrates that the proposed framework can transfer information from historical shock episodes to a new target event in a reproducible and modular way. The same package workflow can be adapted to alternative donor structures, additional pooling rules, or other financial and macroeconomic time series, making the framework flexible for a broad class of postshock forecasting problems.

7 Empirical Application: IYG Volatility around the 2016 U.S. Election

7.1 Empirical setting and evaluation target

This section illustrates the use of `SynthVolForecast()` on a real-data volatility forecasting problem. The target episode is the 2016 U.S. election, and the target asset is IYG, a financial-sector ETF. The goal is to evaluate whether donor post-shock volatility behavior from related historical events can improve a one-step-ahead post-shock variance forecast.

Because the latent conditional variance is not directly observed in real financial data, evaluation is based on the next-day squared return of the target series. The forecasts reported below are variance forecasts, while the visualization uses a square-root volatility-proxy scale for readability.

7.2 Candidate donor pools

The experiment uses the package data object `iyg_onestep_input`, which contains three ready-to-use donor-pool specifications: `all`, `restricted`, and `elections_only`.

```
## [1] "all"            "restricted"     "elections_only"
```

We compare three analyst-specified donor-pool choices. The `all` pool includes all candidate donor episodes and serves as a broad benchmark. The `restricted` pool keeps the 2012 Election, Brexit Poll Released, and Brexit episodes, which are intended to be more comparable political-uncertainty shocks. The `elections_only` pool uses only previous U.S. presidential elections, providing a same-event-type benchmark.

Table 7: Donor-pool specifications for the IYG volatility application.

Donor_Pool	Donor_Episodes
All donors	2004 Election, 2008 Election, 2012 Election, Brexit Announced, Brexit Poll Released, Brexit
Restricted	2012 Election, Brexit Poll Released, Brexit
Elections only	2004 Election, 2008 Election, 2012 Election

The donor-pool definitions are treated as analyst-specified inputs rather than being selected ex post based on forecast accuracy. This design allows the application to evaluate how different substantive choices about donor comparability affect the resulting volatility adjustment.

7.3 Package workflow

For each donor-pool specification, the packaged object provides the complete input required by `SynthVolForecast()`: the target and donor return series, the donor covariate series, the target covariates used for DBW matching, and the aligned shock-time metadata. The same function call is then applied to each donor pool.

7.4 Forecast comparison across donor-pool specifications

The observed next-day squared return is used as the realized-variance proxy for evaluating the one-step-ahead variance forecast.

Table 8: One-step post-shock variance forecasts for IYG under alternative donor-pool specifications.

Pool	DBW	Equal	AE, DBW	AE, equal
Restricted	0.001516	0.001310	0.000235	0.000440
Elections only	0.000233	0.000516	0.001517	0.001234
All donors	0.000778	0.000897	0.000972	0.000854

The common unadjusted forecast is 0.000068, and the observed squared-return proxy is 0.001750. The unadjusted GARCH forecast is much smaller than the realized-variance proxy, suggesting that the pre-shock target volatility model alone does not fully capture the post-shock volatility response. Donor-based adjustment substantially raises the one-step variance forecast. Among the three donor-pool specifications, the restricted pool produces the closest DBW-adjusted forecast to the realized variance proxy. The elections-only pool under-adjusts, while the full donor pool improves relative to the raw forecast but is less accurate than the restricted pool.

The point forecasts do not establish reliable coefficient inference. In the all-donor fit, 4 of 6 donor shock standard errors are non-finite. Centered matching features also generate rank-deficiency warnings. These diagnostics are retained in the executable checks; the convergence filter does not

require finite standard errors. Equal weighting is closer to the observed proxy in the elections-only and all-donor specifications. This single noisy proxy does not establish a general ranking of weighting methods.

The table reports forecasts on the variance scale. The following figure plots the same forecast-day comparison on a square-root volatility-proxy scale, together with the local absolute-return path of the target episode.

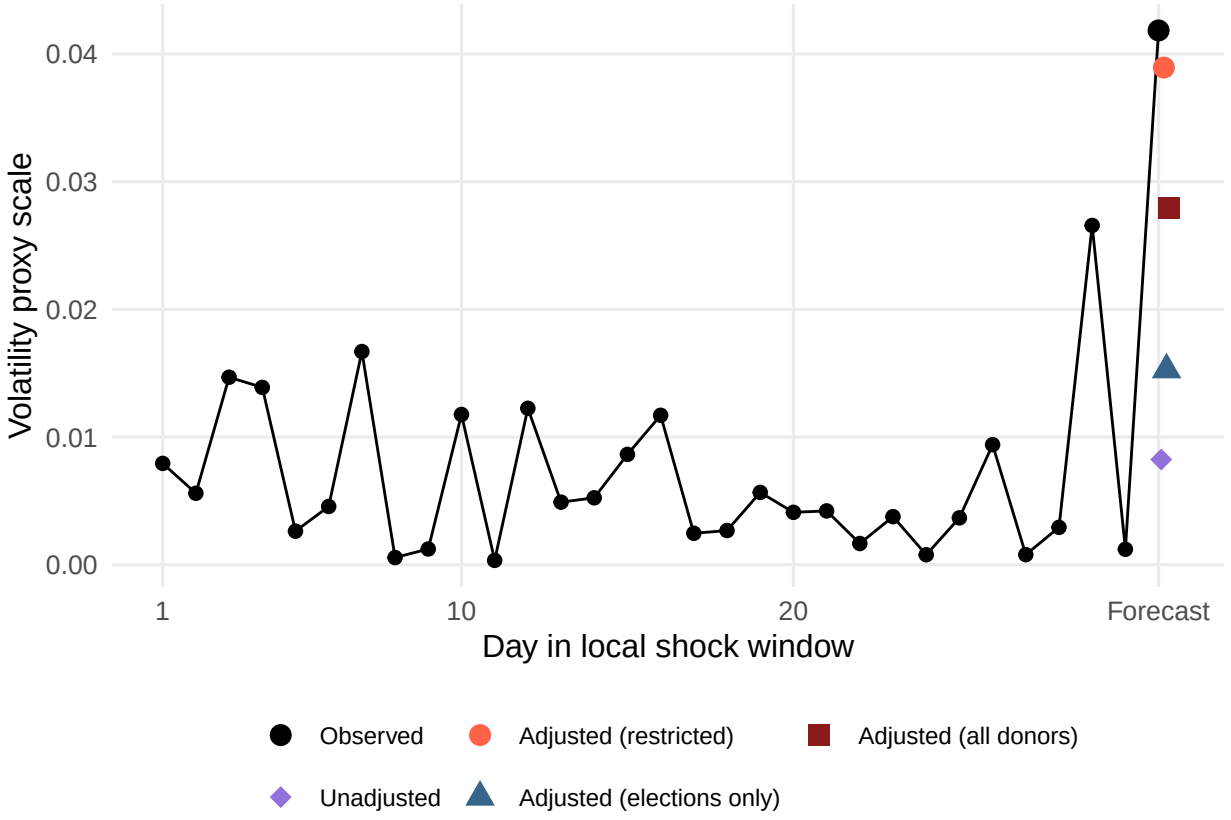


Figure 4: IYG local absolute returns and one-step volatility-proxy forecasts. Forecast markers refer to one evaluation observation; horizontal displacement is for display only.

7.5 Discussion

Taken together, the IYG results suggest that donor-pool composition matters primarily through the magnitude of the implied post-shock volatility uplift. All three donor-pool specifications produce adjusted forecasts above the unadjusted GARCH baseline, indicating that donor information consistently contributes a positive short-horizon volatility correction.

At the same time, the results show that a broader donor pool is not automatically preferable. The full donor pool includes more historical episodes, but this can dilute the adjustment when less comparable episodes receive positive weight. The elections-only pool is transparent from an event-type perspective, but it does not generate a sufficiently large volatility adjustment in this case. The restricted pool balances event comparability and broader political-uncertainty relevance, and it produces the closest DBW-adjusted forecast to the realized variance proxy.

This application complements the simulation study by showing that the same donor-based variance-adjustment logic can be applied to a real financial volatility setting using the packaged `iyg_onestep_input` object. It also illustrates that the software is useful not only for producing forecasts after a donor pool has been specified, but also for comparing how different substantive donor-pool choices affect the resulting post-shock volatility adjustment.

8 Apple empirical application

8.1 Standard package workflow with historical control

The main Apple illustration now uses standard S1: a one-step `SynthPrediction()` forecast of the 11 November 2020 close using the 25 observations from 7 October through 10 November. It is a software adaptation of the historical episodes in Lin and Eck (2021), not an exact reproduction of their delayed-response experiment. The three donor effect dates are retained without claiming independently established homogeneous release mechanisms.

The historical point control is aligned to the response on 9 October (raw target row three), and its forecast-time value is zero. Donor boundaries are one observation before their effect dates; each donor is fitted through that one-period effect window. GSPC and IRX enter ten-observation boundary-aligned DBW matching but not the response models. Centering and scaling are enabled, with no PCA or penalty. Automatic target fitting supplies the baseline.

```
data("aapl_s1_input", package = "postshock")
apple_input <- aapl_s1_input
apple_fit <- SynthPrediction(
  Y_series_list = apple_input$Y_series_list,
  covariates_series_list = apple_input$covariates_series_list,
  shock_time_vec = apple_input$shock_time_vec,
  shock_length_vec = apple_input$shock_length_vec, k = 1,
  use_dbw = TRUE, dbw_indices = 1:2, covariate_indices = NULL,
  days_before_shocktime_vec = 10L, dbw_center = TRUE, dbw_scale = TRUE,
  control_shocks = apple_input$control_shocks,
  arima_order = NULL, seasonal = FALSE, user_ic_choice = "aicc",
  stepwise = TRUE, approximation = TRUE,
  penalty_lambda = 0, penalty_normchoice = "l1"
)
```

Table 9: Apple S1: 11 November 2020 forecasts using information through 10 November.

Forecast	Value	Absolute error
Realized close	119.490	–
Unadjusted baseline	116.050	3.440
DBW-adjusted	117.127	2.363
Equal-weight adjusted	116.928	2.562

A separate target refit in the shared script verifies regression with ARIMA(1,0,0) errors and reproduces the raw package forecast; the returned metadata do not themselves retain this target order. The DBW and equal-weight adjustments are 1.077111 and 0.878302. Both reduce this baseline's underprediction, but substantial error remains. A single retrospective comparison does not establish general predictive superiority or causal confounding removal.

Table 10: Standard S1 donor coefficients and balancing weights.

Effect_date	Effect	Weight
2017-06-07	0.425272	0.583683
2018-10-31	0.218635	0.000000
2019-06-04	1.991000	0.416317

DBW status is convergence. The four-episode, 20-feature matching design produces a rank-deficiency warning after centering (rank 3). This is retained in the executable diagnostics and is not evidence of precisely identified weights. The point forecasts and aggregate adjustments match the audited S1 values at the recorded precision.

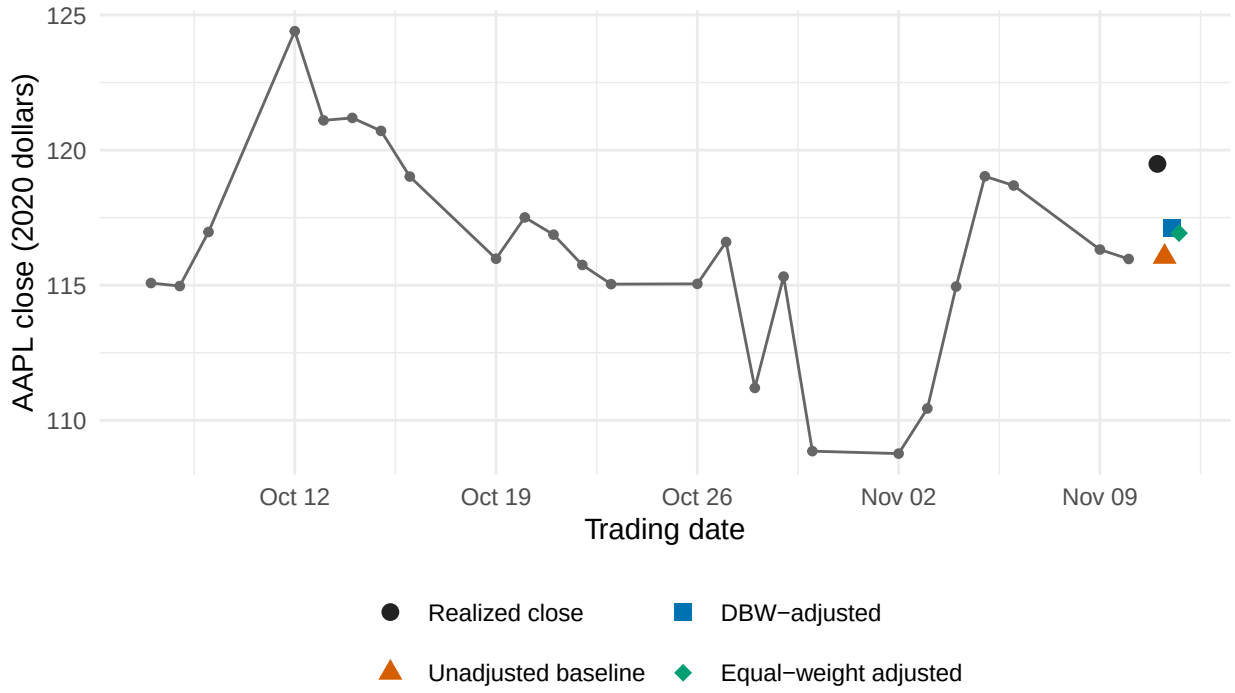


Figure 5: Apple S1 target history and one-step forecasts for 11 November 2020. All comparison markers refer to the same date; horizontal displacement is for display only.

8.2 Modular and delayed-response sensitivity analyses

The supplementary *Apple analyses* document provides the full timing checks, equations, and executable tables for the following fixed specifications. They are kept separate from the main standard

workflow.

- **M:** external OLS with lagged AAPL, GSPC, IRX, and a response-aligned 9 October control, using the same S1 donor adjustment.
- **S1 without control:** remove only the historical target control; donor effects and DBW weights remain unchanged.
- **S1, second horizon:** `k = 2` from 10 November, with the same scalar adjustment at both forecast horizons.
- **S2:** update through 11 November and shift the fixed 25-observation window to forecast 12 November.
- **S3:** use the 24 available observations through 9 November and the standard shared-scalar two-step interface.
- **S4:** application-level delayed-response composition from 9 November. Fixed OLS AR(3)/AR(1) models forecast GSPC/IRX for 10 November; the target baseline recursively forecasts the intermediate response. Full-window donor OLS coefficients and newly estimated effect-horizon DBW weights supply an adjustment only at the second step.

Table 11: Fixed Apple sensitivity specifications; suffix denotes horizon.

Specification	Raw	DBW	Equal	Raw_AE	DBW_AE
M/1	119.978	121.055	120.856	0.488	1.565
S1/2	116.104	117.181	116.982	3.106	2.029
S1_no_control/1	116.031	117.108	116.909	3.459	2.382
S2/1	118.614	119.652	119.493	0.596	0.442
S3/2	116.282	117.375	117.160	3.208	2.115
S4/2	118.878	119.631	120.151	0.612	0.141

M's baseline is already above the observed value, so its positive donor adjustments increase point error. This illustrates baseline/adjustment compatibility rather than software failure. S4 instead changes both timing and donor estimation; it is not native horizon-specific package functionality. The available 24-row target window, control-date adaptation, and retrospective data vintage also preclude calling S4 an exact Lin–Eck reproduction.

The supplementary diagnostics separate baseline differences, control sensitivity, donor-model changes, and supplied versus estimated weights. All tables reuse the shared executable logic, with no copied forecast constants. The former inconsistent modular results and separate automatic control benchmark are superseded and are not current main results.

9 Conclusion

This technical report developed a modular postshock forecasting framework for time series with structural breaks in both mean and volatility. The framework combines donor-based adjustment, synthetic weighting, and flexible extensions for more complex empirical settings.

The simulation studies show that donor-based adjustment improves postshock forecasting relative to unadjusted benchmarks, and that the DBW mechanism is especially useful when donor episodes differ in their pre-shock covariate environments. In the volatility setting, the framework also extends naturally to postshock variance prediction through GARCHX-type modeling.

The IYG volatility application further shows that the same donor-based adjustment logic can be used for real-data variance forecasting. In that example, the restricted donor pool produces the closest one-step variance forecast to the realized variance proxy, highlighting that donor-pool composition matters for volatility adjustment.

The empirical case studies highlight two complementary forms of flexibility. In the ConocoPhillips application, the framework improves upon the unadjusted benchmark and can be further enriched through the MultiPool extension, which groups donor episodes into economically interpretable channels. In the Apple example, the control-shock formulation shows that the same forecasting core can also be applied when the observed postshock response is interpreted in the presence of an additional background component.

Taken together, these results show that the package is not limited to a single fixed application template. Instead, it supports a reproducible and modular workflow that can be adapted to alternative donor structures, different economic mechanisms, and richer interpretations of postshock behavior. Possible future work includes broader real-data volatility applications, additional pooling schemes, and further extensions of the control-shock framework.

10 References

Kilian, Lutz (2009). "Not All Oil Price Shocks Are Alike: Disentangling Demand and Supply Shocks in the Crude Oil Market." American Economic Review, 99(3), 1053–1069.

Lin, Jilei, and Daniel J. Eck (2021). "Minimizing Post-Shock Forecasting Error through Aggregation of Outside Information." International Journal of Forecasting, 37(4), 1710–1727. https://doi.org/10.1016/j.ijforecast.2021.03.010

Sucarrat, Genaro (2021). "garchx: Flexible and Robust GARCH-X Modelling." The R Journal, 13(1), 276–291.