Benchmarking Time-Series Cross Sectional Methods: Synthetic Data Approach

Introduction

Measuring the causal impact of an intervention has long been a central interest for econometricians, and has more recently become a focus in industry, where large scale experiments are frequently run to inform strategic decisions. In this world, the data often contain a large time series component on the outcome of interest, with few relevant covariates, staggered treatment times, and many more control/donor units than treated. Typical empirical applications rely on panel models such as the Synthetic Control Method - henceforth SCM - (Abadie and Gardeazabal 2003; Abadie, Diamond, and Hainmueller 2010, 2015) which impute the unobserved potential outcome for the treated units and compute the Average Treatment Effect on the Treated (ATT) as the average difference between observed and counterfactual outcomes.

The original formulation of SCM predicted counterfactual outcomes as a convex combination of donor units, with weights determined by goodness-of-fit on lagged outcome and auxiliary covariates. However, a number of alternatives have since been proposed, including the introduction of time weights as well as unit weights in Arkhangelsky et al. (2018) SCDID, or Ben-Michael, Feller, and Rothstein (2018)'s bias corrected Augmented Synthetic Control Method (ASCM). More generally, researchers have suggested numerous ways to optimally impute counterfactual outcomes, ranging from Matrix Completion (MC) methods common in computer science (Athey et al. 2017), modeling the outcome as composed of latent factors and estimating these time-varying factors and individual specific loadings (Bai 2009; Gobillon and Magnac 2016; Xu 2017), or using Bayesian Structural Time Series models, as in CausalImpact (Scott and Varian 2014; Broderson et al. 2015).

While this expansive and expanding toolkit provides practitioners with an abundance of choice, it can be hard for practicioners to determine which method is most suitable for their specific purpose. In this paper, our goal is to provide a systematic empirical comparison of these methods as well as an R package intended to guide users to methodologies most suited to their data. Our Monte Carlo comparison relies on a common synthetic DGP framework nesting selection into treatment (random assignment, selection on observables, selection on unobservables), overlap in treated and control time series, heterogeneity in treatment impact and decay, and heterogeneity in the auto-correlation of outcomes. Because the number of potential models is quite large, we focus on a subset primarily based on existing package infrastructure in R – namely CausalImpact, Gsynth, Matrix Completion, and SCDID.

Several papers provide a starting point for our analysis, most notably Samartsidis et al. (2019), who provide a synthesis of the literature on many of these estimation methods and compare their results in an empirical example. Their focus is primarily on describing the various modeling approaches, including uncertainty estimates and inference, drawing connections between the approaches, and describing diagnostics researchers should abide by to determine whether a method is appropriate for the given context. We build off of this work by focusing instead on the empirical comparisons across Monte Carlo simulations, emphasizing a number of distinct particularities in the DGP and how these affect performance across each approach.

Gobillon and Magnac (2016) conduct Monte Carlo experiments in a similar envirionment, comparing several variations of Bai (2009)'s IFE model[1], as well as SCM and standard Difference-in-differences. They conclude that each method is unbiased in their baseline case with random assignment to treatment, but upon introducing

[1]These include standard IFE to compute counterfactual, IFE with a treatment dummy, IFE with counterfactuals based on matches from estimated factor loadings, IFE with counterfactual constrained to SCM weights based on factor loadings

correlations between loadings and treatment assignment, all methods become severely biased except for standard IFE and IFE with a treatment dummy[2].

In a similar vein, Xu (2017) reports results comparing IFE and SCM to their proposed GSC (Gsynth) estimator, which first estimates latent factors using donor units, then estimates factor loadings for each treated observation, and ultimately combines the two to compute counterfactual. The Monte Carlo experiments, using a modified version of the DGP in Bai (2009) and Gobillon and Magnac (2016), suggest that the Gsynth method is less biased than traditional difference-in-differences when there exist unobservable, decomposable time-varying confounders, and is less biased than IFE under heterogeneous treatment effects. Lastly, they show that Gsynth tends to be more efficient than SCM, and demonstrate the robustness of these results by recomputing the metrics using cross-validation to estimate the number of factors instead of assuming the correct model specification.

Finally, Gardeazabal and Vega-Bayo (2017) sample from existing data as well as their synthetic DGP to compare SCM to a panel data approach (IFE) by Hsiao, Ching, and Wan (2012) under different circumstances. Interestingly, they explore how robust the methodologies are to changes in the donor pool – selecting the subsample of donors from those that have positive weights according to SCM or those that are statististically significant under the IFE approach – and conclude that SCM is more robust to these alterations. However, in simulation studies focusing on cases with bad pre-treatment matches (defined by Mean Absolute Error, MAE), the results suggest that SCM is much more biased, while IFE models are unbiased but have large confidence intervals.

Our benchmarking exercise adds to this existing literature in several ways. First, we provide a more expansive set of comparisons relative to existing MC studies by introducing generalized models on a common DGP; SCDID generalizates both Difference-in-differences and SCM; the Gsynth package (Xu 2017) provides implementations for one or two-way fixed effects, IFE using EM (Gobillon and Magnac 2016), Matrix Completion (Athey et al. 2017), as well as the Gsynth algorithm; CausalImpact covers a large number of time series models under the umbrella of BSTS (including ARIMA models). Second, the flexibility of our DGP allows us to examine each approach under a number of interesting cases. Third, inspired by an expansive literature on forecasting time-series, we provide a simple characterization of the data using time-series features such as ACF and entropy, and analyze whether differences in these features are predictive of differences in methodology performance (Kang, Hyndman, and Smith-Miles 2017). Fourth, borrowing from machine learning, we study whether ensembles can provide performance boosts in this context (Hastie, Tibshirani, and Friedman 2009; Athey et al. 2019).

The rest of the paper proceeds as follows. Section @ref(sec:methods) provides a brief overview of each of the methods we analyze herein, with appropriate references for further information. Section @ref(sec:dgp) provides an annotated walkthrough our DGP, which serves as the basis for all of our MC simulations.

Methods Overview

Notation

Throughout the paper, we refer to individuals as $i \in (1, 2, \ldots, N)$ and time periods as $t \in (1, 2, \ldots, T)$. A subset of the N units are treated at varying times, and once treated, remain so for the duration. Units belong to one of two subsets: treated (N_{tr}) and control (N_c) units, such that $N = N_{tr} + N_c$. The total number of time periods, T, can be similarly decomposed into the pre-treatment periods, $t \in (1, 2, \ldots, T_0)$, and post-treatment periods ($t \in (T_0 + 1, T_0 + 2, \ldots, T)$) given the time $T_0 + 1$ that unit i is first treated. Let D_{it} represent whether unit i received treatment in time t, so $D_{it} = 1$ if $i \in N_{tr}$ and $t \geq T_{0_i}$, with $D_{it} = 0$ otherwise. Potential outcomes for unit i at time t are $Y_{it}(0), Y_{it}(1)$ for the outcome under control and treatment respectively, while observed outcomes are

$$Y_{it} = \begin{cases} Y_{it}(0) & D_{it} = 0 \\ Y_{it}(1) & D_{it} = 1, \end{cases}$$

[2]Difference-in-differences is also unbiased here, but the authors demonstrate that this is an artifact of their main DGP.

which we can rewrite in the Rubin causal framework as $Y_{it} = (1 - D_{it})Y_{it}(0) + D_{it}Y_{it}(1)$ (Rubin 1974).

Our goal is to estimate the Average Treatment on the Treated (ATT), defined as $ATT_t = \tau_t = E[Y_t(1|D_{it} = 1) - Y_t(0|D_{it} = 1)]$. Because we only observe one potential outcome, our methods impute the missing variable in their various ways to form our estimate,

$$\hat{ATT}_t = \sum_{i \in N_{tr}} Y_{it}(1) - \hat{Y}_{it}(0).$$

We next describe the ways in which this imputation is computed.

Generalized Synthetic Control (Gsynth)

Proposed by Xu (2017), Gsynth adopts Interactive Fixed Effects (IFE) models to impute $\hat{Y}_{it}(0)$. In particular, they posit that the outcome takes the functional form

$$Y_{it} = \tau_{it} D_{it} + x'_{it}\beta + \lambda'_i f_t + \epsilon_{it},$$

which is a standard representation of IFE models. Here, we have a $(k \times 1)$ vector of observable covariates, x_{it}, a $(k \times 1)$ vector of unknown parameters, $\beta = [\beta_1, \ldots, \beta_k]'$, an $(r \times 1)$ vector of unobserved, time-varying, common factors $f_t = [f_{1t}, \ldots, f_{rt}]'$ and an $(r \times 1)$ vector of their individual specific, time constant loadings $\lambda_i = [\lambda_{i1}, \ldots, \lambda_{ir}]'$. This factor-loading formulation nests additive time and unit fixed effects, as well as autoregressive components (see Gobillon and Magnac 2016, and @xu_2017 for more information).

Because the (potentially heterogeneous) treatment effect enters as an additively separable term, Xu (2017) suggests the following procedure. First, estimate the unknown parameters on the control units ($\hat{\beta}, \hat{\lambda_i}, \hat{f}_t$, $i \in N_c$). This is done following an algorithm proposed by Bai (2009) for estimating latent factors.[3] Second, using $(\hat{\beta}, \hat{f}_t)$, estimate the loadings for each treated unit by minimizing the MSE of predicted outcomes in the *pre-treatment* periods only, $t \in (1, \ldots, T_0), i \in N_{tr}$. Formally,

$$\hat{\lambda}_i = \underset{\tilde{\lambda}}{\operatorname{argmin}} (Y_i^0 - X_i^0\hat{\beta} - \hat{F}^0\tilde{\lambda}_i)'(Y_i^0 - X_i^0\hat{\beta} - \hat{F}^0\tilde{\lambda}_i),$$

where super-script 0 denotes the stacked vector of pre-treatment periods for the relevant variable, with $i \in N_{tr}$. Third, impute the counterfactual outcome for treated units in the post-treatment periods by predicting the outcome using $\hat{\beta}, \hat{f}_t$ from step 1 and $\hat{\lambda_i}$ from step 2:

$$\hat{Y}_{it}(0) = x'_{it}\hat{\beta} + \hat{\lambda}'_i\hat{f}_t, \quad i \in N_{tr}, t > T_0 + 1.$$

The resulting estimator for the ATT at time $t > T_0$ is $\hat{ATT}_t = \frac{1}{N_{tr}} \sum_{i \in N_{tr}} Y_{it}(1) - \hat{Y}_{it}(0)$.

According to Xu (2017), this method requires several assumptions for causal identification. Given the functional form, we also require the treated and control units are affected by the same r factors, with r fixed over the duration. Next, a strict exogeneity assumption is introduced – $\epsilon_{it} \perp\!\!\!\perp D_{js}, x_{js}, \lambda_j, f_s \ \forall i, j, t, s$. While this requires that the error terms for any unit in any period is independent of treatment assignment, observed covariates, and the factors/loadings, it nevertheless allows for the treatment indicator to be correlated with observable or unobservable covariates. Additional assumptions on the error term (weak serial dependence) and regularity conditions are imposed for the consistency of β, see Xu (2017) for details.

This method has several beneficial features, including the generality of IFE model (factor/loadings combination nest models such as two-way additive fixed effects, and even AR processes) as well as a key computational advantage: unlike SCM methods, the estimator only needs to run once on the control panel to obtain the factors and coefficients, making it faster than competing estimators. As for drawbacks, Xu (2017) suggests that there can be bias problems with few pre-treatment periods (incidental parameter problem), and it relies heavily on the modeling assumptions (the estimator extrapolates as if the factors for treated and control units were the same, when they may not be).

[3] This algorithm requires additional constraints to identify β, f, λ: first, that factors are normalized and second that they are orthogonal to each other; details in Bai and Ng (2002); Bai (2009).

SCDID

In their paper, Arkhangelsky et al. (2018) present Difference in Difference and the Synthetic Control Method as falling under a unified framework by reformulating SCM as weighted least squares regression. From there, Synthetic Difference in Differences is a natural extension of the SCM which includes both time-weights and unit-weights (in a multiplicative fashion), as well as unit fixed effects. The key advantage of these additional features is that they yield improved bias properties and a form of double robustness – the estimator is consistent if either the outcome model is correctly specified or weights are well chosen.

Their example takes the simplest case – only unit N is treated at time T – from which they characterize the SCM and DiD estimators in a regression form:

$$(\hat{\mu}, \hat{\alpha}, \hat{\beta}, \hat{\tau}^{did}) = \underset{\mu,\alpha,\beta,\tau}{\operatorname{argmin}} \sum_i \sum_t (Y_{it} - \mu - \alpha_i - \beta_t - D_{it}\tau)^2$$

$$(\hat{\mu}, \hat{\beta}, \hat{\tau}^{scm}) = \underset{\mu,\beta,\tau}{\operatorname{argmin}} \sum_i \sum_t (Y_{it} - \mu - \beta_t - D_{it}\tau)^2 \hat{\omega}_i^{scm}.$$

This allows us to easily see that, while the DiD approach has both time and unit fixed effects, the SCM approach uses only time fixed effects with unit specific weights. Hence, Arkhangelsky et al. (2018) propose extending SCM to include these "missing" unit fixed effects, as well as a modification to the weights

$$(\hat{\mu}, \hat{\alpha}, \hat{\beta}, \hat{\tau}^{scdid}) = \underset{\mu,\alpha,\beta,\tau}{\operatorname{argmin}} \sum_i \sum_t (Y_{it} - \mu - \alpha_i - \beta_t - D_{it}\tau)^2 \hat{\omega}_i \hat{\lambda}_t.$$

Another way of thinking about these methods is how they impute the missing outcome. DID imputes

$$\hat{Y}_{NT}^{did}(0) = \bar{Y}_{nc,t} + (\bar{Y}_{N,t} - \bar{Y}_{n,t}) + (\bar{Y}_{n,T} - \bar{Y}_{n,t}),$$

adjusting the mean of the control units pre-treatment outcome by the stable differences over time between control and treated units, and the stable differences over time within the control group. Synthetic control goes one step further by finding weights for each control unit such that this weighted combination of control units tracks the treated unit through time; imputation is now

$$\hat{Y}_{NT}^{scm}(0) = \frac{1}{T-1} \sum_{i \in N_c} \sum_{t<T} \hat{\omega}^{scm} Y_{it} + \sum_{i \in N_c} \hat{\omega}^{scm} \left(Y_{iT} - \frac{1}{T-1} \sum_{t<T} Y_{it} \right)$$

. However, this method omits the second bias correction term present in DID – the stable difference between the treated unit and the weighted control units in the pre-treatment periods. SCDID extends SCM by adding this bias adjustment on pre-treatment differences using weight $\hat{\lambda}^s c$, so that imputation in SCDID follows

$$\hat{Y}_{NT}^{scdid}(0) = \hat{Y}_{NT}^{scm}(0) + \sum_{t<T} \hat{\lambda}_t^{scm} \left(Y_{Nt} - \sum_{i \in N_c} \hat{\omega}_i^{scm} Y_{it} \right).$$

More generally, if the outcomes are generated by $Y_{it} = L_{it} + D_{it}\tau + \epsilon_{it}$, the authors propose parameterizing L_{it} as $g(\theta)$ (examples include IFE/Latent Factor models, two-way fixed effects) and estimating the parameters by solving

$$\operatorname{argmin}_\theta \sum_i \sum_t (Y_{it} - g(\theta)_{it} - \tau D_{it}) \hat{\omega}_i \hat{\lambda}_t.$$

The weight terms, $\hat{\omega}_i, \hat{\lambda}_t$ emphasize the focus of getting $g(\theta)$ right locally near the treated unit. These weights are assumed non-negative and sum to one (separately within each group). Unit weights are estimated following the standard SCM paradigm,

$$\hat{\omega}_i = \operatorname{argmin}_\omega \sum_{t<T} (\sum_{i \in N_c} \omega_i Y_{it} - Y_{Nt})^2 + \gamma ||\omega||_2,$$

with the L_2 norm imposed on the weights to introduce sparsity (which is helpful for interpretation as well as the asymptotic properties). Time weights – the novelty of this appraoch compared to SCM – are estimated as

$$\hat{\lambda}_t = \text{argmin}_\lambda \sum_{i \in N_c} (\sum_{t<T} \lambda_t Y_{it} - Y_{iT})^2.$$

In terms of assumptions, Arkhangelsky et al. (2018) provide a qualitative description: the estimator assumes that weights exist such that the average treated unit and the weighted average of the controls satisfy a parallel trends assumption for the averaged post-treatment period and the weighted average of the pre-treatment periods. This serves to relax the DiD assumption of parallel trends for all time periods and units. More formally, the main assumption is that there exists a determinstc $\mathbf{L}$ such $Y_{it} = L_{it} + D_{it}\tau + \epsilon_{it}$, where $\epsilon_{it} \sim \mathcal{N}(0, \sigma^2)$ for each (i, t).

As described in their framework, one of they key advantages of this approach is the double robustness type property, which yields improved bias properties relative to SCM and DiD. Moreover, the introduction of the time weights helps to capture time series patterns in the data – for instance, a quarterly seasonality could be captured by increasing the importance of the outcome four periods ago. However, this approach maintains the SCM convexity assumptions on the weights, namely that they be positive and sum to one, a potentially unnecessary assumption which precludes the use of information from negatively correlated time series.[4]

Causal Impact

Introduced by Broderson et al. (2015), Causal Imapct is a fully Bayesian time-series methodology for estimating the causal effect in the settings we consider. Specifically, this approach treats the post-treatment counterfactual outcomes (of the treated units) as unobservable random variables and estimates these by using information from 1) patterns identified in the pre-treatment time series of the treated unit, 2) relationships between control series and the treated series, and 3) covariate series, if applicable. Because the time series approach is build off of Bayesian Structural Time Series (Scott and Varian 2014), models can become quite complicated and are very flexible, with the ability to nest all ARIMA models.

As described in Broderson et al. (2015), the authors differ from the DID and SCM framework in their explicit choice to select donor time series based purely on their explanatory power. Specifically, they do not restrict weights on these variables to be a convex combination, and constrain model complexity via regularization priors. This distinction, as well as the ability to learn from the treated series of the treated unit, are major benefits of this appraoch. However, this reliance on the time-series structure necessitates more pre-treatment periods in order to provide value. Lastly, there does not seem to be a way to compute the Causal Impact for more than one treated series at a time, slowing computation dramatically.

Matrix Completion

Matrix Completion is a method common among computer scientists and statisticians in which missing matrix entries are imputed under a typical assumption of MCAR (missing completely at random). Athey et al. (2017) make a slight alteration to these methods by allowing data to be missing according to particular patterns, and show how these patterns relate to the literature on SCM and matching with unconfoundedness. To see this, take $\mathbf{Y}$ as our $N \times T$ matrix of potential outcomes without treatment (typical element $Y_{it}(0)$); all units have an observed value in the pre-treatment period but only control units have an observed value in the post-treatment periods.

$$Y_{match} = \begin{pmatrix} \checkmark & \checkmark & \dots & \checkmark & \checkmark \\ \checkmark & \checkmark & \dots & \checkmark & \checkmark \\ \checkmark & \checkmark & \dots & \checkmark & ? \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \checkmark & \checkmark & \dots & \checkmark & ? \end{pmatrix}$$

[4] I've seen this mentioned in either the ASCM or the rSCM papers, but want to make sure this is accurate.

$$Y_{scm} = \begin{pmatrix} \checkmark & \checkmark & \dots & \checkmark & \checkmark \\ \checkmark & \checkmark & \dots & \checkmark & \checkmark \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \checkmark & \checkmark & \dots & \checkmark & \checkmark \\ \checkmark & ? & \dots & ? & ? \end{pmatrix}$$

We can think of datasets suited to matching estimators as those like $Y_{matching}$, where we would typically compute a **horizontal** regression by imputing the missing outcomes using a regression of the final period outcomes on lagged outcomes (perhaps weighting the control units that look most like the treated). On the other end of the spectrum, data like Y_{scm} – where we are missing several post-treatment period outcomes for a single treated unit – are typically evaluated with **vertical** regression by imputing missing outcomes based on a regression of the treated unit's pre-treatment outcomes on the controls (essentially finding pre-treatment weights that best predict the treated unit, and extrapolating through time). Matrix completion can then be thought of as a framework that discerns which of these two (or other) shapes the Y matrix takes and, given this pattern, imputes the missing outcomes in the most appropriate way.

Formally, the MC problem without covariates is set up as $Y = L^* + \epsilon$, where each matrix is $(N \times T)$, $E[\epsilon|L^*] = 0$, and the goal is to estimate L^*. The key assumption is Matrix Unconfoundedness, which is to say that $D \perp\!\!\!\perp \epsilon|L$, where D is the matrix with element $D_{it} \in (0, 1)$ indicating whether the potential outcome $Y_{it}(0)$ is observed. The proposed estimator in Athey et al (2018) is

$$\hat{L} = \underset{L}{\text{argmin}} \frac{1}{|\mathcal{O}|} \sum_{(i,t)\in\mathcal{O}} (Y_{it} - L_{it})^2 + \lambda_L ||L||_*,$$

where $\mathcal{O}$ is a collections of observed indices (i, j), with Nuclear Norm

$$||L||_* = \sum_i \sigma_i(L)$$

, where $\sigma_i(L)$ are the singular values of L. To solve this problem, the authors first define a few additional terms.

First, we let

$$P_{\mathcal{O}}(A)_{it} = \begin{cases} A_{it} & (i,t) \in \mathcal{O} \\ 0 & (i,t) \notin \mathcal{O}. \end{cases} \qquad P^{\perp}_{\mathcal{O}}(A)_{it} = \begin{cases} A_{it} & (i,t) \notin \mathcal{O} \\ 0 & (i,t) \in \mathcal{O}. \end{cases}$$

Next, define a matrix shrinkage operator

$$\text{shrink}_\lambda(A) = S\tilde{\Sigma}R'$$

, where the right hand side represents the SVD of A, and shrinkage is applied via $\tilde{\Sigma}$ by replacing $\sigma_i(A) = \max(\sigma_i(A) - \lambda, 0)$. Initialize $L_1(\lambda, \mathcal{O}) = P_{\mathcal{O}}(Y)$ (or the 0 matrix), and compute

$$L_{k+1}(\lambda, \mathcal{O}) = \text{shrink}_{\frac{\lambda|\mathcal{O}|}{2}} \left(P_{\mathcal{O}}(Y) + P^{\perp}_{\mathcal{O}}(L_k(\lambda)) \right)$$

, until the sequence converges. The limiting matrix serves as the estimator for a given λ, which is itself estimated via cross validation.[5]

The authors then go on to present a theorem demonstrating that the SCM and IFE approaches can be viewed as a MC method based on matrix factorization with the same objective function but different restrictions on the factors (see Athey et al. 2017 for details). Thus, this method is quite general and adaptive to the structure of the missing data problem at hand. Practically speaking, the success of this method depends on whether L can be well approximated by a low-rank matrix. Moreover, unlike the SCDID and Gsynth appraoches, the method takes advantage of the pre-treatment period data for the treated units, allowing for extrapolation based on observed structure.[6]

[5] Take K random subsets of observed indices – restricting the fraction of observed data in these subsets to be equal to the full data, then find the limiting matrix for a sequence of lambdas, comparing the performance by lambda and K.

[6] What are some drawbacks besides "it doesn't work"??

DGP

In this section, we outline the DGP used as the basis of our benchmarking exercises. We begin by giving a highlighting the key features of our DGP in a high-level overview, and then go into code and details in a separate subsection for interested readers.

Key Features

Our data is generating from the following factor augmented autoregressive process of order one:

$$lnY_{it}(0) = \alpha_i + \rho_i lnY_{i,t-1}(0) + F_t\lambda_i' + \epsilon_{it}$$
$$lnY_{it}(1) = lnY_{it}(0) + D_{it}\tau_{it}.$$

Each unit has an individual specific intercept (α_i), auto-correlation coefficient (ρ_i), as well as factor loadings, (λ_i, dimension $1 \times r$, where r is the number of unobserved factors) which scale unobserved, time-varying factors (F_t, dimension $1 \times r$ for a given t). The noise term, ϵ_{it} is iid $\mathcal{N} \sim (0, 0.5^2)$. Finally, treatment impact is modeled as a shock to counterfactual outcomes, with our binary indicator D_{it} encoding treatment status and τ_{it} representing the impact of treatment for unit i at time t.

We allow for a number of treatment assignment mechanisms, such as random assignment ($E(\alpha_i|D_{it} = 1) = E(\alpha_i|D_{it} = 0)$ as well as for ρ_i, λ_i), selection on observables (D_{it} is correlated with at least one of $\alpha_i, \rho_i, \lambda_i$), and selection on unobservables.[7] Extreme observations can also be made more frequent by tweaking an overlap parameter, which shifts the distribution of covariates[8] for a fraction of observations, and then randomly assigns treatment.[9]

Furthermore, we allow for varying degrees of heterogeneity in both treatment impact and decay. Our treatment effect $\tau_{it} = \omega_i * \delta_{it}$ flexibly covers many cases, including impact decay by setting (for example) $\delta_{it} = \delta_i^{t-T_{0i}}$ – where δ_i itself can display heterogeneity – as well as impact heterogeneity via ω_i.

Implementation Details

We walk through the details of our DGP code for ease of use. The DGP has a core function that is called with all of the arguments, and then sends much of the work to various helper functions to help keep the flow clear. The following chunk of code details the inputs to the DGP; after verifying the function inputs are feasible, we determine the length of the total time period (which depends on input dates and time frequency) as well as the time when treatment is allowed to begin. We then send the relevant subset of inputs to a helper function which creates the time-invariant (unit level) variables, another relevant subset to a helper creating the time-varying variables (factors), bring the two together to form our unit by time grid, generate the counterfactual outcome process, and add the treatment to the counterfactual. Note that "factor_synthetic_dgp()" generates

$$lnY_{it}^*(0) = \alpha_i + \rho_i lnY_{i,t-1}^*(0) + F_t\lambda_i'$$

which is then supplemented with random (for now, iid) noise to create bootstrap datasets for bias, inference, and coverage estimates.

```
pacman::p_load(dplyr, furrr, tibble, tidyr, stats, lubridate,
               glue, truncnorm,MASS)
#TODO(alexdkellogg): allow dgp to be covariates only or factors only
#TODO(alexdkellogg): allow num_periods directly rather than dates
```

[7] For now, the outcome is actually not effected by additional observables or unobservables. However, these variables have been created, selection on them is modeled, and the outcome can be made to depend on them by modeling $\alpha_i = a_i + \beta_{obs}X_{1i} + \beta_{unobs}X_{2i}$.

[8] For now, we just shift observable and unobservable X, which do not interact with Y, though we should easily be able to shift ρ, λ.

[9] I can't tell if this is helpful above and beyond traditional selection.

```r
factor_synthetic_dgp<-function(num_entries=2000,
                               date_start="2017-01-01",
                               first_treat="2018-07-01",
                               date_end="2020-01-01",
                               freq=c("daily", "weekly", "monthly"),
                               prop_treated=0.4,
                               treat_impact_mean=0.1,
                               treat_impact_sd=0.1,
                               treat_decay_mean=0.7,
                               treat_decay_sd=1,
                               selection=c("random", "observables",
                                           "unobservables"),
                               rho=0.9,
                               rho_scale=0.2,
                               rho_shift=0,
                               cov_overlap_scale=0,
                               num_factors=3,
                               loading_scale=0,
                               intercept_scale=0,
                               conditional_impact_het=-0,
                               rescale_y_mean=2.5e4,
                               seed=19){
  #rescale_y_max=5e8
  #=0,

  #Generates tibble of long form panel data using a factor-augmented AR 1
  #Each row is time period x unit unique combination.

  #Args
  #num_entries: number of units to be generated
  #date_start: string "yyyy-mm-dd" input for the first simulated date
  #first_treat: string "yyyy-mm-dd" input first treatment period.
  #date_end:: string "yyy-mm-dd" input for the last simulated date
  #freq: string indicating time unit, either "daily", "weekly", "monthly"
  #prop_treated: proportion of entries that should receive treatment
  #treat_impact_mean: initial period treatment impact mean, drawn from truncated
  #   normal distribution centered here. The end points of the dist are [0,0.25]
  #   by default, but shift to [a,b] where b=mean+0.25 if the mean is larger
  #   than 0.25 (max of 1)  or a=mean-0.25 if mean is below 0.
  #treat_impact_sd: standard deviation of the truncated normal mean impact.
  #treat_decay_mean: initial period treatment decay mean, drawn from truncated
  #   normal distribution centered here. The end points of the dist are [0,0.9]
  #   by default, but shift to [0,1+eps] if mean>0.9. This allows for units to
  #   have no decay (value of 1). Propagates as mean**(post_treat_period).
  #treat_impact_sd: standard deviation of the truncated normal decay factor
  #selection= string in "random", "observables", "unobservables" dictating
  #   treatment assignment mechanism.
  #rho: mean of truncated normal distribution of the autocorrelation of outcome,
  #   with bounds [0, 0.995].
  #rho_scale: standard deviation of truncated normal for the autocorrelation.
  #rho_shift: multiplier on the mean rho for control units,
  #   overall mean stay below 1 (rho*rho_shift<1).
  #cov_overlap_scale: (-1,1) shifts distribution of covariates for a fraction
```

```
#   (prop_treated) of x variables. A value of 1 shifts the distribution for
#   treatment up on all x's, whereas -1 shifts up the distribution for donors.
#num_factors: number (3+) of time-varying, unobserved factors to simulate
#loading_scale: (-1,1) shift in factor distribution, -1 shifts loadings up
#   for control units, positive values shift loadings distribution up for
#   treated units.
#intercept_scale: (-1,1) shifts the mean of a truncated normal distribution
#   for control unit intercepts: >0 shifts the mean down (treatment is larger)
#   and <0 shift control mean above the treatment mean.
#conditional_impact_het: constant added to the treatment impact for top 25%
#   and subtracted from bottom 25% of counterfactual y at time 1.
#rescale_y_mean: number representing the target mean of exp(counterfactual)



#Output
#Long form tibble with columns for observed, potential treated, and
#  potential untreated outcomes, treatment time, and
#  the relevant x variables (loadings, intercept, observables).




#Match the arguments to valid entries
set.seed(seed)
if (missing(selection)) {
  selection <- "random"
} else{
  selection <- match.arg(selection)
}

if (missing(freq)) {
  freq <- "monthly"
} else{
  freq <- match.arg(freq)
}
#require 10% min in both treat and control
stopifnot(prop_treated >= 0.1 & prop_treated <= 0.9)
#require cov_overlap_scale to be between -1 (shift treat down) and 1
stopifnot(cov_overlap_scale <= 1 & cov_overlap_scale >= -1)
#require 3 factors of more
stopifnot(num_factors>2)
#require less than 100% TE
stopifnot(treat_impact_mean<1)

#given the dates and frequency, identify total number of periods
num_periods=time_interval_calculator(start_t=date_start, end_t=date_end,
                                     freq_t=freq)
#Store how long after start date the treatment begins
treat_start_int=time_interval_calculator(start_t=date_start,
                                         end_t=first_treat, freq_t=freq)
#Stop if there are too few pre treat periods
stopifnot(treat_start_int < 0.8*num_periods)
```

```
#
#assign covariates and treatment given selection and overlap
synth_data_unit=unit_level_simulation(n_inp=num_entries,
                              type_inp=selection,
                              cov_overlap_inp=cov_overlap_scale,
                              loading_scale_inp=loading_scale,
                              num_factors_inp=num_factors,
                              int_scale_inp=intercept_scale,
                              rho_inp=rho,
                              rho_scale_inp=rho_scale,
                              rho_shift_inp=rho_shift,
                              prop_treated_inp=prop_treated,
                              treat_start=treat_start_int,
                              num_periods_inp=num_periods,
                              impact_mean_inp=treat_impact_mean,
                              impact_sd_inp=treat_impact_sd,
                              decay_mean_inp=treat_decay_mean,
                              decay_sd_inp=treat_decay_sd)

#next, work on time varying characteristics
synth_data_factors=generate_factors(num_factors_inp=num_factors,
                                    num_periods_inp=num_periods,
                                    num_entry_inp=num_entries,
                                    date_start_inp=date_start,
                                    date_end_inp=date_end,
                                    freq_inp=freq)

#Next, we generate both potential outcomes
#Option for factor only (y=factor*loadings), Random Effect only
# (y=xB), or both.
#Then, generate the treatment impact
unit_time_grid <-tidyr::expand_grid(entry = seq_len(num_entries),
                                    time = seq_len(num_periods))

synth_data_full=unit_time_grid %>%
  dplyr::left_join(synth_data_unit,by=c("entry")) %>%
  dplyr::left_join(synth_data_factors,by=c("time"))

#generate the counterfactual outcomes
synth_data_full=generate_counterfactual(synth_data_full,
                                        num_periods_inp=num_periods,
                                        rescale_y=rescale_y_mean)


#generate the per period impact (taking acocunt of decay)
synth_data_full=generate_treat_impact(data_inp=synth_data_full,
                                      cond_impact_inp=
                                        conditional_impact_het)

return(synth_data_full)
```

```
}

noisify_draw<-function(data_inp, seed,log_output=T, sig_y=1){
  set.seed(seed)
  eps=stats::rnorm(nrow(data_inp), sd=sig_y)
  return(data_inp %>%
    dplyr::mutate(y=exp(y+eps)*(1- log_output)+log_output*(y+eps),
                  y0=exp(y0+eps)*(1- log_output)+log_output*(y0+eps),
                  y1=exp(y1+eps)*(1- log_output)+log_output*(y1+eps)))
}
```

Individual Level Work

Similar to the overall process, the individual level helper function itself splits off tasks to more specific functions. In this case, there are several processes that need handling: treatment period assignment, treatment status assignment, covariate creation, and treatment impact assignment. The flow for these processes can be seen in the code below. These helper function aggregate the covariates, treatment assignment, and treatment start date into an N row tibble. These in hand, the process concludes by drawing individual specific treatment impact and decay rate: ω_i, δ_i. $\omega_i \sim TN_{[a,b]}(\mu_\omega, s_\omega^2)$, where μ_ω, s_ω come from user input, allowing for impact to be concentrated or dispersed. One note, mentioned before, is that the bounds of the truncated normal distribution default to $[a, b] = [0, 0.25]$, but can shift to $[0, \min(\mu_\omega + 0.25, 1)]$ if $\mu_\omega > 0.25$, or $[\mu_\omega - 0.25, 0.25]$ if $\mu_\omega < 0$. Decay rate is also drawn from a truncated normal, $\delta_i \sim TN_{[0,b]}(\mu_\delta, s_\delta^2)$, where $b = 0.9$ by default, but moves to $b = 1 + \epsilon$ to allow for no decay if the user wishes.

```
unit_level_simulation <- function(n_inp,
                                  type_inp,
                                  cov_overlap_inp,
                                  loading_scale_inp,
                                  num_factors_inp,
                                  int_scale_inp,
                                  rho_inp,
                                  rho_scale_inp,
                                  rho_shift_inp,
                                  prop_treated_inp,
                                  treat_start,
                                  num_periods_inp,
                                  impact_mean_inp,
                                  impact_sd_inp,
                                  decay_mean_inp,
                                  decay_sd_inp){
  #Gather data that is time invariant
  #First, call to assign treatment, which generates covariates
  unit_level_tib=assign_treat(n_inp=n_inp,
              type_inp=type_inp,
              cov_overlap_inp=cov_overlap_inp,
              loading_scale_inp=loading_scale_inp,
              num_factors_inp=num_factors_inp,
              rho_inp=rho_inp,
              rho_scale_inp=rho_scale_inp,
              rho_shift_inp=rho_shift_inp,
              int_scale_inp= int_scale_inp,
              prop_treated_inp=prop_treated_inp)

  #Merge in the treatment period assignment
```

```
  unit_level_tib=assign_treat_time(unit_level_tib, treat_start, num_periods_inp)
  #allow wiggle room for the TE without getting NA
  impact_ub=ifelse(impact_mean_inp<0.25,0.25,
                   min(1, impact_mean_inp+(0.25)) )
  impact_lb=ifelse(impact_mean_inp<=0,impact_mean_inp-0.25, 0)
  decay_ub=ifelse(decay_mean_inp>0.9,1.000000001, 0.9)
  #assign treatment effect impact and decay parameters per unit
  unit_level_tib=unit_level_tib %>%
    dplyr::mutate(
      treat_impact=truncnorm::rtruncnorm(n=dplyr::n(), a=impact_lb,b=impact_ub,
                                         mean=impact_mean_inp, sd=impact_sd_inp),
      treat_decay=truncnorm::rtruncnorm(n=dplyr::n(), a=0,b=decay_ub,
                                        mean=decay_mean_inp, sd=decay_sd_inp))

  return(unit_level_tib)
}
```

We start with the simplest process, which designates treatment time. Treatment time is dictated by a geometric distribution as well as user input for when treatment may start; the geometric distribution models the count of independent Bernoulli trials up to the first success when the probability of each trial is p. Depending on the number of post treatment periods available to us, we choose $p \in (0.25, 0.2, 0.15, 0.1)$ to help ensure that we have a reasonable amount of post treatment available for the majority of our observations (See Figure @ref(fig:treat-start-time) for an example).

```
assign_treat_time<-function(treat_tib_inp, treat_start, num_periods_inp){
  #Geometric distribution parameter selected based on time
  #Goal is to have most treatment assigned before end date
  geom_prob=dplyr::case_when(num_periods_inp-treat_start<=20~0.25,
                             num_periods_inp-treat_start<=30~0.2,
                             num_periods_inp-treat_start<=40~0.15,
                             TRUE~0.1 )
  #draw treatment period as first date plus geometric random variable
  return(treat_tib_inp %>%
           dplyr::mutate(
             treatment_period=treat_start+
               stats::rgeom(dplyr::n(), geom_prob)) %>%
           dplyr::group_by(entry) %>%
           dplyr::mutate(treatment_period=
                           min(treatment_period,num_periods_inp)) %>%
           dplyr::ungroup())

}
```

We next discuss treatment assignment; because we want to handle selection and covariate overlap, our treatment assignment function actually handles much of the work at the individual level. Specifically, it first generates covariates with a fraction (the proportion treated) of the covariates coming from a distribution shifted according to a covariate overlap input (can be shifted up or down).[10] Next, in the case of random assignment, a random uniform variable is generated independent on covariates, and the top proportion treated fraction are assigned treatment, and are passed on to another function that assigns individual specific intercepts, loadings, and autocorrelation parameter.

[10]Currently, these covariates only affect selection into treatment and are fixed in number and type. We could allow for generalization here by allowing the user to input the desired number of X's, and then randomly selecting a subset (and storing the name) for use in the selection process?

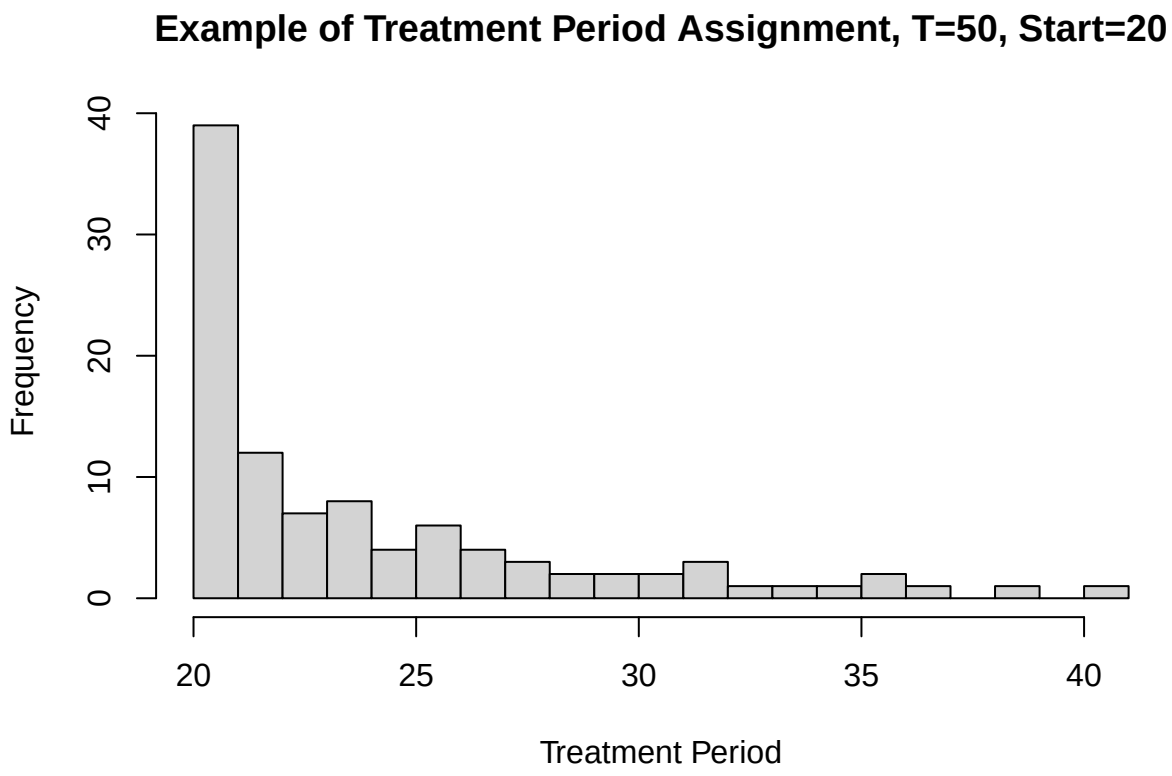


Figure 1: Assignment Example

```
#TODO(alexdkellogg): Generalize to allow for user inputted number of xs
gen_covariates<-function(n_inp, cov_overlap_inp, frac_shifted){
  #Generate several covariates, both observed and unobserved.
  #Shift means according to overlap_inp and frac_shifted

  #first, create a set of correlated variables, mean zero, var/covar Sigma
  Sigma <- matrix(c(16 ,  4,  -4.8,
                    4  , 25,    9,
                    -4.8,  9,    9),3,3)
  corr_xs= MASS::mvrnorm(n = n_inp, rep(0, 3), Sigma)
  colnames(corr_xs)=c("obs_mvnorm1", "unobs_mvnorm", "obs_mvnorm2")

  return(
    x_tib=tibble::tibble(
      entry = seq_len(n_inp),
      to_shift = as.numeric(dplyr::percent_rank(
        runif(n = n_inp,min = 0,max = 1)
                          )>1-frac_shifted)) %>%
    cbind(corr_xs) %>%
    dplyr::group_by(to_shift) %>%
    dplyr::mutate(
      obs_beta=stats::rbeta(n = dplyr::n(),
                            shape1 = 6,
                            shape2 = 5+3*(to_shift*-(cov_overlap_inp))),
      obs_binom=stats::rbinom(n = dplyr::n(),
                            size = 15,
```

```
                          prob = 0.5+0.25*(to_shift*(cov_overlap_inp))),
      obs_norm=stats::rnorm(n = dplyr::n(),
                            mean = 0+5*(to_shift*(cov_overlap_inp)),
                            sd = 10),
      unobs_beta=stats::rbeta(n = dplyr::n(),
                          shape1 = 6,
                          shape2 = 5+3*(to_shift*-(cov_overlap_inp))),
      ubobs_binom=stats::rbinom(n = dplyr::n(),
                            size = 15,
                            prob = 0.5+0.25*(to_shift*(cov_overlap_inp))),
      ubobs_norm=stats::rnorm(n = dplyr::n(),
                          mean = 0+5*(to_shift*(cov_overlap_inp)),
                          sd = 10) ) %>%
      dplyr::ungroup() %>%
      dplyr::select(-to_shift)
  )

}
```

Intercepts are distributed according to $\alpha_i \sim TN_{[5,10]}(\mu_\alpha, 0.7^2)$, where $\mu_\alpha = 7.5 - (1 - D_i) * het_{int}$ is a function of treatment status. Individual autocorrelation is drawn from a truncated normal as well, $\rho_i \sim TN_{[0,0.995]}(\mu_\rho * (1 - D_i) * het_\rho, s_\rho^2)$, with $\mu_\rho, het_\rho, s_\rho^2$ all coming from user input. This allows for homogeneity in ρ, heterogeneity alike for treated and untreated units, or differential means by treatment status. Finally, r loadings – according to the number of factors the user requested – are drawn from $\lambda_i \sim \beta(2 - (1 - D_i)het_{load}, 2 - D_i het_{load})$, so that for $het_{load} > 0$, loadings are drawn from a distribution shifted towards one for treated and shifted toward zero for control units (opposite if below 0).

```
generate_loadings <- function(treat_tib_inp, loading_scale_inp,num_factors_inp,
                              int_scale_inp,rho_inp, rho_scale_inp, rho_shift_inp){

  loadings_mat=matrix(0, nrow=nrow(treat_tib_inp), ncol=num_factors_inp)
  colnames(loadings_mat)=glue::glue("loading{1:num_factors_inp}")


  tib_pre_loadings=treat_tib_inp %>%
    dplyr::group_by(treated) %>%
    dplyr::mutate(
      #intercept=stats::rexp(dplyr::n(), 1+(1-treated)*int_scale_inp),
      intercept=truncnorm::rtruncnorm(n=dplyr::n(), a=5,b=10,
                                      mean=7.5-(1-treated)*int_scale_inp,
                                      sd=0.7),
      #intercept=stats::runif(dplyr::n(), min=5,max=8-(1-treated)*int_scale_inp),
      # autocorr=stats::runif(dplyr::n(),ifelse(treated,rho_inp,
      #                                         rho_inp*rho_scale_inp),
      #                       max=0.95),
      autocorr=truncnorm::rtruncnorm(n=dplyr::n(), a=0,b=0.995,
                                     mean=ifelse(treated,rho_inp,
                                                 rho_inp*rho_shift_inp),
                                     sd=rho_scale_inp)) %>%
    dplyr::bind_cols(tibble::as_tibble(loadings_mat)) %>%
    dplyr::ungroup()
  return(
    tib_pre_loadings %>%
      dplyr::group_by(treated) %>%
```

```
      dplyr::mutate(
        dplyr::across(tidyselect::starts_with("loading"),
        .fns=list(load=~stats::rbeta(dplyr::n(), 2 -(1-treated)*(loading_scale_inp) ,
                    2 - (treated)*(loading_scale_inp)) ),
        .names ="{col}")
      ) %>% dplyr::ungroup()

  )


}
```

If instead a selection mechanism was desired, the relevant covariates (observed or unobserved) are selected, multiplied by a random normal coefficient vector, and mapped into the probability space by the logistic function. To add additional noise to the selection, a random uniform is added to the probability after this transformation (which results in scores in [0,2]), and the top proportion_treated are assigned treatment. Given this treatment assignment, we send the data off to a helper function which creates the intercept, loadings, and autocorrelation as describe above.[11] An example of shifting the loadings up fro the treated group is demonstrated in Figure @ref(fig:selection-ex-fig).

```
assign_treat<-function( n_inp, type_inp, cov_overlap_inp,
                        loading_scale_inp,rho_inp=rho_inp,
                        rho_scale_inp,rho_shift_inp, num_factors_inp,
                        int_scale_inp, prop_treated_inp){
  #Creates a tibble with treatment assignment and time constant covariates
  #Covariate distribution can differ by treatment depending on cov_overlap_inp,
  #and selection into treatment is modelled.

  #generate covariates with varying levels of overlap
  covariate_tib=gen_covariates(n_inp=n_inp, cov_overlap_inp=cov_overlap_inp,
                               frac_shifted=prop_treated_inp)

  if (type_inp == "random") {
    #Randomly assign treatment to prop_treated_inp
    treat_covar_tib=tibble::tibble(
      entry = seq_len(n_inp),
      treated = as.numeric(dplyr::percent_rank(
        runif(n = n_inp,min = 0,max = 1)
      )>1-prop_treated_inp)) %>%
      inner_join(covariate_tib, by="entry")

    unit_tib=generate_loadings(treat_tib_inp=treat_covar_tib,
                                     loading_scale_inp=loading_scale_inp,
                               num_factors_inp=num_factors_inp,
                                     int_scale_inp=int_scale_inp,
                               rho_inp=rho_inp,
                                     rho_scale_inp=rho_scale_inp,
                               rho_shift_inp = rho_shift_inp)
    return(unit_tib)
  }
  else{
```

[11]Note that, as of now, the selection is not technically governed the loadings/intercept/autocorrelation, but the distribution of these variables can optionally be shifted up or down by treatment status so that treatment assignment is correlated by these variables.

```
    #Assign treatment based on either observable or unobservable xs
    relevant_xs=ifelse(type_inp=="observables", "obs", "unobs")
    #First, create covariates with
    z <- covariate_tib %>%
      dplyr::select(tidyr::starts_with(relevant_xs)) %>%
      as.matrix()

    # combine the variables for each observation to form a predicted score
    # rescale with logistic function to map into the probability space (0,1)
    prob_treat= tibble::tibble(score=
                                 z %*% stats::rnorm(n = ncol(z),mean = 0,sd = 1)) %>%
      dplyr::mutate(prop_score = 1 / (1 + exp(score)))

    #Ad random noise to the score and take the top fraction as treated
    treat_covar_tib= prob_treat %>%
      dplyr::mutate( u = runif(n = n_inp, min = 0, max = 1),
                     p_new = prop_score+u,
                     treated=as.numeric(dplyr::percent_rank(p_new)>
                                          1-prop_treated_inp),
                     entry = seq_len(n_inp)) %>%
      dplyr::select(entry, treated) %>%
      inner_join(covariate_tib, by="entry")

    unit_tib=generate_loadings(treat_tib_inp=treat_covar_tib,
                               loading_scale_inp=loading_scale_inp,
                               num_factors_inp=num_factors_inp,
                               int_scale_inp=int_scale_inp,
                               rho_inp=rho_inp,
                               rho_scale_inp=rho_scale_inp,
                               rho_shift_inp = rho_shift_inp)

    return(unit_tib)
  }
}
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

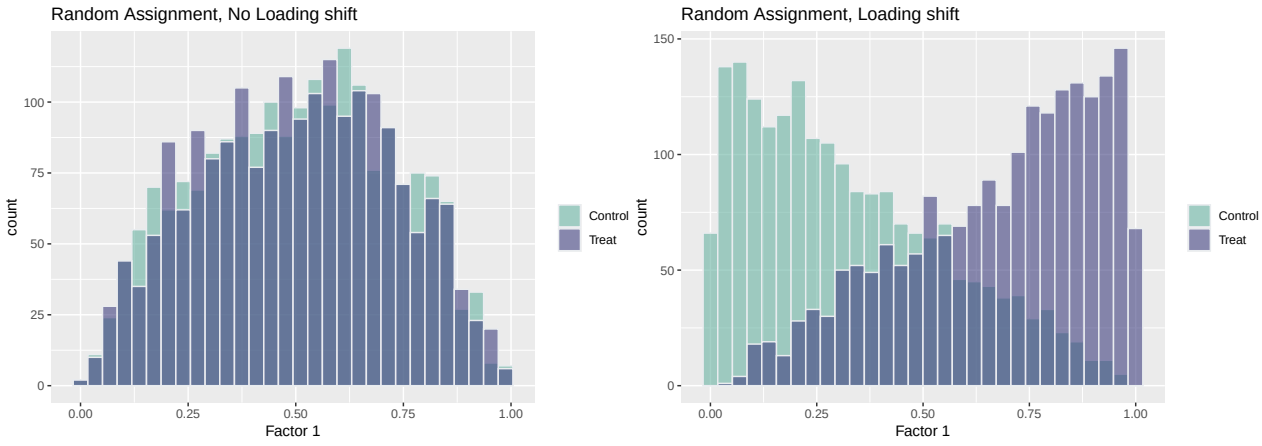


Figure 2: Overlap Example

Time-Varying Factors

The next component of our DGP is the creation of our factors, which are common across all individuals but vary over time. These are loosely modeled on existing data, from which we estimated 4 factors resembling the following processes:

$$
\begin{aligned}
F_{1t} &= t/T + \nu_{1t} \\
F_{2t} &= \rho_{F_2} F_{2,t-1} + \alpha_q + \nu_{2t} \\
F_{3t} &= \rho_{F_3} F_{3,t-1} + \alpha_m + \nu_{3t} \\
F_{4t} &= \rho_{F_4} F_{4,t-1} + \alpha_r + \nu_{4t}.
\end{aligned}
$$

At a conceptual level, the first factor represents a time trend, the second captures a seasonal component every quarter, the third a seasonal component every month, and the fourth (and beyond) represent random fixed effects at varying intervals. Mathematically, $\rho_{Fj} = 0.2 \; \forall j$ so that the autoregressive component of each factor is fixed at 0.2. The shocks, $\alpha_j \sim U[-1, 1]$, are iid and drawn each quarter ($j = q$), month ($j = m$), or stochastically – with the number of shocks drawn from a discrete $U[1, 13]$ and their location drawn from discrete $U[1, 51]$.[12] Idiosyncratic noise for each factor is drawn from $\nu_j \overset{\text{iid}}{\sim} \mathcal{N}(0, 0.1^2)$. In building the AR(1) process for each factor, we allow a burn in period of 500.

```
generate_factors<-function(num_factors_inp,
                           num_periods_inp, num_entry_inp,
                           date_start_inp, date_end_inp, freq_inp){

  #generate factors from an AR 1 process
  factor_mat <-matrix(0, nrow=num_periods_inp,ncol = num_factors_inp )
  colnames(factor_mat) <- glue::glue("factor{1:num_factors_inp}")
  #ar model description -- AR 1 with auto correlation and sd inputs
  ar_model=list(order=c(1,0,0), ar=0.2)
  #TODO(alexdkellogg): check with AP if shocks trend over time, assumed fixed
  quarter_effects=tibble::tibble(q_shock=stats::runif(4, -1, 1),
                                 quarter_num=seq_len(4))
  month_effects=tibble::tibble(m_shock=stats::runif(12, -1, 1),
                               month_num=seq_len(12))


  #combine the zero matrix of factors with date indicators
  factor_tib=generate_time_grid(date_start_inp=date_start_inp,
                                num_periods_inp=num_periods_inp,
                                freq_inp=freq_inp,
                                num_entry_inp=num_entry_inp) %>%
    dplyr::bind_cols(tibble::as_tibble(factor_mat)) %>%
    dplyr::inner_join(quarter_effects, by="quarter_num") %>%
    dplyr::inner_join(month_effects, by="month_num")


  #add first factor, period/total + noise
  factor_tib=factor_tib %>%
    dplyr::mutate(factor1=time/dplyr::n()+
                    stats::rnorm(dplyr::n(),mean=0,sd=0.1)) %>%
    dplyr::group_by(quarter_num) %>%
```

[12] In using the R-package "lubirdate", we generate a time grid with information on the quarter and month for each row, which we use to assign the shocks at the proper intervals.

```
    dplyr::mutate(
      factor2=stats::arima.sim(model=ar_model, n=dplyr::n(),
                               innov = q_shock+stats::rnorm(dplyr::n(),
                                                            sd=0.1),
                               n.start = 500)) %>%
    dplyr::ungroup() %>%
    dplyr::group_by(month_num) %>%
    dplyr::mutate(
      factor3=stats::arima.sim(model=ar_model, n=dplyr::n(),
                               innov = m_shock+stats::rnorm(dplyr::n(),
                                                            sd=0.1),
                               n.start = 500)) %>%
    dplyr::ungroup()
  if(num_factors_inp>3){
    #this process works for one factor. Want to lapply to all columns >4
    extra_factors_names=setdiff(
      names(factor_tib %>% dplyr::select(tidyselect::contains("factor"))),
      c(glue::glue("factor{1:3}"))
      )
    extra_factor_tib=purrr::map(.x=extra_factors_names,
                                .f=~add_extra_factors(factor_tib=factor_tib,
                                  num_factors_inp=num_factors_inp,
                                  num_periods_inp=num_periods_inp,
                                  ar_model_inp=ar_model, col_in = .x)) %>%
      dplyr::bind_cols()

    factor_tib=factor_tib %>%
      dplyr::select(-tidyselect::all_of(extra_factors_names)) %>%
      dplyr::bind_cols(extra_factor_tib)
  }

  return(factor_tib %>% dplyr::select(-tidyselect::contains("shock")))

}

generate_time_grid <- function(date_start_inp,num_periods_inp,
                               freq_inp, num_entry_inp){
  period_dates=switch(freq_inp,
                      "daily"=format(lubridate::ymd(date_start_inp)+
                                       lubridate::days(0:(num_periods_inp-1))),
                      "weekly"=format(lubridate::ymd(date_start_inp)+
                                        lubridate::weeks(0:(num_periods_inp-1))),
                      "monthly"=format(lubridate::ymd(date_start_inp)+
                                         months(0:(num_periods_inp-1)))
  )

  #Identidy the relevant components of the date (day/week/etc)
  date_info_tib=switch(freq_inp,
                       "daily" = tibble::tibble(
                         time=seq_len(num_periods_inp),
                         date_t=period_dates,
                         day_num=lubridate::day(period_dates),
                         week_num=lubridate::week(period_dates),
```

```
                            month_num=lubridate::month(period_dates),
                            quarter_num=lubridate::quarter(period_dates),
                            year_num=lubridate::year(period_dates)),

                          "weekly"=tibble::tibble(
                            time=seq_len(num_periods_inp),
                            date_t=period_dates,
                            week_num=lubridate::week(period_dates),
                            month_num=lubridate::month(period_dates),
                            quarter_num=lubridate::quarter(period_dates),
                            year_num=lubridate::year(period_dates)),
                          "monthly"=tibble::tibble(
                            time=seq_len(num_periods_inp),
                            date_t=period_dates,
                            month_num=lubridate::month(period_dates),
                            quarter_num=lubridate::quarter(period_dates),
                            year_num=lubridate::year(period_dates))
                          )

  return( date_info_tib )
}
```

Our DGP supports a minimum of 3 factors so that the trend and seasonal components do a reasonable job mimicking the existing data we estimated.[13] More than 3 factors can also be supported, where each factor above 3 follows the process defined for F_4 above (all independent from one another). An example with 4 factors is provided in Figure @ref(fig:factors-ex-fig).

```
add_extra_factors<-function(factor_tib,col_in,num_factors_inp,
                            num_periods_inp,
                             ar_model_inp){

  #compute the number shocks and their respective locations
  num_shocks=sample(1:13,1)
  shock_locs=c(0,sort(sample(1:52, size =num_shocks, replace = F )),52)
  extra_shocks=stats::runif(n=num_shocks+1, min=-1, max=1)
  shock_seq=rep(rep(extra_shocks, diff(shock_locs)), length.out=num_periods_inp)




  shockXwalk=tibble::tibble(time=seq_len(num_periods_inp),
                            e_shocks=shock_seq)
  factor_tib=factor_tib %>%
    dplyr::left_join(shockXwalk, by="time") %>%
    dplyr::mutate(
    !!as.name(col_in):=
      stats::arima.sim(model=ar_model_inp, n=dplyr::n(),
                       innov = e_shocks+stats::rnorm(dplyr::n(),
                                                     sd=0.1),
                       n.start = 500))

  return(factor_tib %>% dplyr::select(tidyselect::all_of(col_in)))
```

[13] If we would like, we can change this to be a purely random effects model, akin to Ignacio's DGP. That is to say, drop the factors and loadings entirely and use only the observed (and unobserved) covariates to model Y, which can still have a trend.

```
}
```

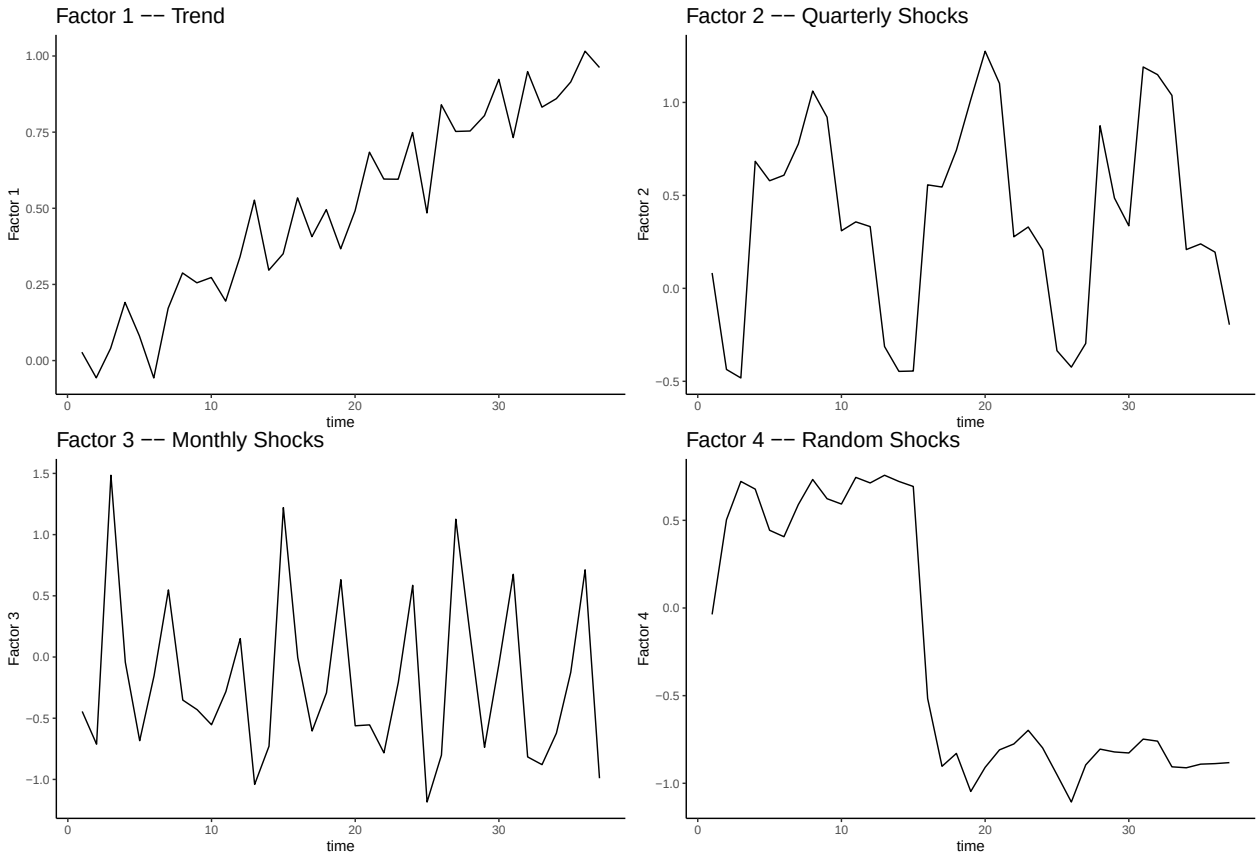


Figure 3: Factors Example

Computing Counterfactuals

The final step in the DGP computes the counterfactual outcome process and adds the treatment impact at each time period to round out our two potential outcomes as well as our observed outcome. As defined above, the outcome follows an AR(1) process with innovations governed by the time-varying factors and their individual specific loadings, as well as iid noise. Thus, we first compute the $T \times 1$ dimensional column $F_t\lambda_i'$ for each individual in parallel. With these computed, we simulate the AR(1) process for the counterfactual, where the process is once again governed by an individual specific parameter – ρ_i – allowing for parallelization of the code.[14] With this series generated, we add the intercept by entry before applying treatment effects, and rescale the potential (untreated) outcome so that the distribution resembles its empirical counterpart.

```
generate_counterfactual<-function(data_inp,num_periods_inp, rescale_y){

  outcome_series=data_inp %>%
    dplyr::select(time, entry, intercept, autocorr,
                  tidyselect::matches("loading|factor"))

  computed_factor_vec=compute_factor_loadings(outcome_series)

  outcome_series=outcome_series %>%
```

[14] As before, we allow a burn in period of 500.

```
    dplyr::select(-tidyselect::matches("loading|factor")) %>%
    dplyr::mutate(factor_loadings=computed_factor_vec)

  outcome_ar=compute_outcome_process(outcome_series,num_periods_inp)

  xvars=data_inp %>%
    dplyr::distinct(entry, .keep_all=T) %>%
    dplyr::select(tidyselect::matches("obs")) %>%
    as.matrix()

  x_beta=xvars %*% stats::rnorm(n=ncol(xvars), sd=0.5) %>%
    as.vector()

  x_betaXwalk=tibble::tibble(entry=seq_len(length(x_beta)),
                             xbeta=x_beta)
  outcome_series=outcome_series %>%
    dplyr::left_join(x_betaXwalk, by="entry") %>%
    dplyr::mutate(factor_loadings=computed_factor_vec,
                  indiv_component=intercept+xbeta,
                  y0=outcome_ar+indiv_component)

  outcome_series_rescaled=outcome_series %>%
    dplyr::mutate(de_sd_y0=(y0/sd(y0)),
                  rescaled_y=de_sd_y0-(mean(de_sd_y0)-log(rescale_y)),
                  y0=1.125*rescaled_y) %>%
    dplyr::select(time, entry, factor_loadings, y0)

  return(data_inp %>%
           dplyr::inner_join(outcome_series_rescaled, by=c("time", "entry")) %>%
           dplyr::select(time, entry, treated, treatment_period,y0,
                         factor_loadings, dplyr::everything())
         )


}


compute_factor_loadings <- function(data_inp){
  #create a list of factor loadings split by individual
  loadings_vec_list=data_inp %>%
    dplyr::select(entry, tidyselect::contains("loading")) %>%
    dplyr::distinct(entry, .keep_all=T) %>%
    dplyr::group_split(entry, .keep=F) %>%
    lapply(as.matrix)
  #create a list of factor matrices split by individual
  factor_mat_list=data_inp %>%
    dplyr::select(entry, tidyselect::contains("factor")) %>%
    dplyr::group_split(entry, .keep=F) %>%
    lapply(as.matrix)
  #Compute the matrix product of loadings and factors for each individual
  return(furrr::future_map2(.x=loadings_vec_list,.y=factor_mat_list,
                                        .f=~(.y)%*%t(.x) ) %>%
    unlist())
```

```
}

#TODO(alexdkellogg): check with AP about 0 noise AR for y
compute_outcome_process<-function(data_inp,num_periods_inp){
  #create a list of AR models, with individual specific noise and autocorr
  ar_param_inp=data_inp %>%
    dplyr::select(entry, autocorr) %>%
    dplyr::distinct(entry, .keep_all=T) %>%
    dplyr::group_split(entry, .keep=F) %>%
    lapply(function(x){ list(order=c(1,0,0), ar=x[[1]])} )


  innov_list=data_inp %>%
    dplyr::select(entry, tidyselect::contains("loading")) %>%
    dplyr::group_split(entry, .keep=F)   %>%
    lapply(as.matrix)


  return(furrr::future_map2(.x=ar_param_inp, .y=innov_list,
                            .f=~stats::arima.sim(model=.x,
                                                 n=num_periods_inp,
                                                 n.start = 500,
                                                 innov = .y)) %>%
    unlist())

}
```

The final step is then to add the treatment impact. Throughout our DGP, we have assigned each unit a hypothetical treatment assignment time, treatment impact, and treatment decay rate; this allows us to generate the treatment impact for all units, regardless of their true treatment status, and store each of the two potential outcomes in our final tibble. For each time period, we compute the individual specific treatment impact as the product of the original impact and the amount of decay by that particular point in time. We also allow for a conditional treatment impact boost (or drop) to the top 25% and conditional drop (or boost) to the bottom 25% of observations based on their counterfactual outcome at $t = 1$. For time before the treatment period, this is simply 0, yielding our counterfactual. For the post-treatment periods, we can add this quantity to the potential (untreated) outcome to form our potential (treated) outcome for each individual, and finally, assign the observed outcome following the standard framework. Lastly, we take the exponent of each of the three outcome variables, as outcome distributions are roughly lognormal, and this shifts the scale to the empirical benchmark.

```
generate_treat_impact<-function(data_inp=synth_data_full,
                                cond_impact_inp){
  #determine which observations get a conditional treatment boost
  data_inp=data_inp %>%
    dplyr::left_join(
      data_inp %>%
        dplyr::filter(time==1) %>%
        dplyr::select(entry, y0) %>%
        dplyr::mutate(
          cond_treat_impact=dplyr::case_when(
            y0>quantile(y0,0.75)~cond_impact_inp,
            y0<quantile(y0,0.25)~-cond_impact_inp,
            TRUE~0 )), by="entry")
```

```
  #Define the treatment propogation for each unit and time combo
  data_inp=data_inp %>%
    dplyr::group_by(entry) %>%
    dplyr::mutate(
      post_treat_t=time-treatment_period,
      decay_t=dplyr::case_when(
        post_treat_t<0~0,
        post_treat_t>=0~treat_decay**post_treat_t),
      impact_t=decay_t*(treat_impact+cond_treat_impact),
      ) %>%
    dplyr::ungroup() %>%
    dplyr::select(-c(treat_decay, treat_decay, decay_t, cond_treat_impact))

  #Add the treatment impact to create y1
  return(data_inp %>%
    dplyr::mutate(y1=impact_t+y0,
                  y=treated*y1+(1-treated)*y0) %>%
      dplyr::select(time, entry, treated,
                    post_treat_t, treatment_period, impact_t,
                    y, y0,y1,
                    dplyr::everything()))
}
```

Computation Time

We briefly describe the performance of this DGP and how it scales with increasing number of entries and/or time periods, as well as the desired number of draws. We choose the base case (with default parameters, including only 3 factors) as our point of departure, which takes about 0.72 seconds. From here, quintupling the number of entries (ceteris paribus) increases runtime by about 2.5x, doubling the number of time periods leaves the runtime approximately the same, and quintupling the entries while also doubling time increases runtime by roughly 3.3x. Drawing 100 samples from the baseline DGP takes about 5 seconds when running in parallel on 3 cores, scaling the draws to 500 takes about 5x as long (23 seconds, omitted here), and 500 draws from the N=1000 DGP takes about 120 seconds (omitted here).

```
## DGP Baseline computation time, T=121; N=200: 0.992 sec elapsed

## DGP Baseline computation time, T=121; N=1000: 2.399 sec elapsed

## DGP Baseline computation time, T=241; N=200: 1.012 sec elapsed

## DGP Baseline computation time, T=241; N=1000: 3.331 sec elapsed

## 100 draws from N=200, T=121: 6.127 sec elapsed
```

Preliminary Results of MC Experiments

To compare the relevant methods under our aforementioned DGP, we take a draw of Y^* under the desired parameters and generate M draws of noise – forming M datasets with $Y_m = Y* + \epsilon_m$. Each Y_m provides us N_{tr} individual treatment effect series, over which we jackknife the $\hat{ATT}$ and its $1-\alpha/2$ confidence interval (for a given post-treatment time period).[15] Once we have obtained $\hat{ATT}$ estimates for each of the M draws of noise, we can compute a number of important metrics to compare across methods: 1) coverage of the confidence interval, by counting the percentage of M for which the true ATT lies within the jackknifed

[15] A more complete implementation, which we hope to have available soon, uses Chernozhukov et al (2019)'s exact conformal inference for this type of data instead of the jackknife for the confidence intervals.

bounds; 2) the bias, by applying the jackknife over the M draws on the mean of $\hat{ATT}_m - ATT$; 3) the variance of $\hat{ATT}$, which we jackknife; 4) RMSE, which we compute by jackknifing the mean of $(Y_{it} - Y_{it}^*)^2$ for post treatment t and taking the square root.

We also introduce an ensemble estimator similar to that described in Athey et al (2019). Given the dataset, we create a placebo by identifying the N_{tr} control series that are the best match for our treated units (without replacement and in an $L2$ distance sense); the remaining unmatched controls serve as the placebo donors to our matched placebo treated units. Each of the placebo treated units is assigned a treatment period based on its corresponding treated unit, but no placebo impact is generated (ie regardless of the true treatment effect, there is no impact in this constructed data). We then re-estimate our candidate methods using this constructed data to form predictions in the relevant post-periods, and find the weighted combination of these methods that minimizes the sum of squared residuals. We allow the option for an intercept (included in the examples below) as well as optional constraints on the weights – namely that they should be positive and sum to 1.[16] Finally, we apply these weights to the estimates from the original dataset and define these as our ensemble predictions, with associated metrics following the standard definitions above.

A comprehensive list of DGP variations is included in the appendix, along with graphs of the bias, graphs of overlap and tables depicting the similarity of the time series by treatment, and tables comparing the methods by coverage, RMSE, and bias over the first 4 post-treatment periods. Overall, Gsynth seems to be the clear winner, displaying robustness to variations in the autocorrelation factor, the amount of noise in each time series, selection based on factor-loadings, as well as heterogeneity of both treatment impact and treatment decay. The Matrix Completion method from the Gsynth pacakge performs noticeably worse than the other methods in the presence of a correlation between loadings and treatment assignment.[17].

SCDID and Causal Impact perform similarly across the variations, with slight sensitivity to selection as well as the noisiness of the data for SCDID and low autocorrelation for Causal Impact. These performance sensitivies mostly show up in the variations without treatment effects, and predominantly in the form of overly narrow confidence intervals. Interestingly, these problems do not manifest much in the variations with treatment effects; there, SCDID and Causal Impact tend to be more robust to selection, and often have overly conservative confidence intervals (at least for the first period or two). In addition, SCDID is more likely to have a (slight) negative bias in situations where Gsynth/MC/Causal Impact have positive bias.

References

Abadie, Alberto, Alexis Diamond, and Jens Hainmueller. 2010. "Synthetic Control Methods for Comparative Case Studies: Estimating the Effect of California's Tobacco Control Program." *Journal of the American Statistical Association* 105 (490): 493–505.

———. 2015. "Comparative Politics and the Synthetic Control Method." *American Journal of Political Science* 59 (2): 495–510.

Abadie, Alberto, and Javier Gardeazabal. 2003. "The Economic Costs of Conflict: A Case Study of the Basque Country." *American Economic Review* 93 (1): 112–32.

Arkhangelsky, Dmitry, Susan Athey, David A. Hirshberg, Guido W. Imbens, and Stefan Wager. 2018. "Synthetic Difference in Differences." http://arxiv.org/abs/1812.09970.

Athey, Susan, Mohsen Bayati, Nikolay Doudchenko, Guido Imbens, and Khashayar Khosravi. 2017. "Matrix Completion Methods for Causal Panel Data Models." http://arxiv.org/abs/1710.10251.

Athey, Susan, Mohsen Bayati, Guido Imbens, and Zhaonan Qu. 2019. "Ensemble Methods for Causal Effects in Panel Data Settings." http://arxiv.org/abs/1903.10079.

Bai, Jushan. 2009. "Panel Data Models with Interactive Fixed Effects." *Econometrica* 77: 1229–79.

[16] We currently model these constraints jointly using quadratic programming – thus, you may either enforce both constraints or let both fluctuate.

[17] The stark drop in performance makes me wonder if there is a bug in my implementation – perhaps the default hyperparameters are not adequate?

Bai, Jushan, and Serena Ng. 2002. "Determining the Number of Factors in Approximate Factor Models." *Econometrica* 70: 191–221.

Ben-Michael, Eli, Avi Feller, and Jesse Rothstein. 2018. "The Augmented Synthetic Control Method." http://arxiv.org/abs/1811.04170.

———. 2019. "Synthetic Controls and Weighted Event Studies with Staggered Adoption." http://arxiv.org/abs/1912.03290.

Broderson, Kay H., Fabian Gallusser, Jim Koehler, Nicolas Remy, and Steven L. Scott. 2015. "Inferring Causal Impact Using Bayesian Structural Time-Series Models." *Annals of Applied Statistics* 9 (1): 247–74.

Doudchenko, Nikolay, and Guido W. Imbens. 2016. "Balancing, Regression, Difference-in-Differences and Synthetic Control Methods: A Synthesis." http://arxiv.org/abs/1610.07748.

Gardeazabal, Javier, and Ainhoa Vega-Bayo. 2017. "An Empirical Comparison Between the Synthetic Control Method and Hsiao et Al.'s Panel Data Approach to Program Evaluation." *Journal of Applied Econometrics* 32: 983–1002.

Gobillon, Laurent, and Thierry Magnac. 2016. "Regional Policy Evaluation: Interactive Fixed Effects and Synthetic Controls." *The Review of Economics and Statistics* 98 (3): 535–51.

Hastie, Trevor, Robert Tibshirani, and Jerome H Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Edited by 2. Springer.

Hsiao, Cheng, H. Steve Ching, and Shui Ki Wan. 2012. "A Panel Data Approach for Program Evaluation: Measuring the Benefits of Political and Economic Integration of Honk Kong with Mainland China." *Journal of Applied Econometrics* 27: 705–40.

Imbens, Guido W., and Donald B Rubin. 2015. *Causal Inference in Statistics, Social, and Biomedical Sciences*. Cambridge University Press.

Kang, Yanfei, Rob J Hyndman, and Kate Smith-Miles. 2017. "Visualising Forecasting Algorithm Performance Using Time Series Instance Spaces." *International Journal of Forecasting* 33 (2): 345–58.

Rubin, Donald B. 1974. "Estimating Causal Effects of Treatments in Randomized and Nonrandomized Studies." *Journal of Educational Psychology* 66 (5): 688–701.

Samartsidis, Pantelis, Shaun R Seaman, Anne M Presanis, MAtthew Hickman, and Daniela De Angelis. 2019. "Assessing the Causal Effect of Binary Interventions from Observational Panel Data with Few Treated Units." *Statistical Science* 34 (3): 486–503.

Scott, Steven L., and Hal Varian. 2014. "Predicting the Present with Bayesian Structural Time Series." *International Journal of Mathematical Modelling and Numerical Optimisation* 5: 4–23.

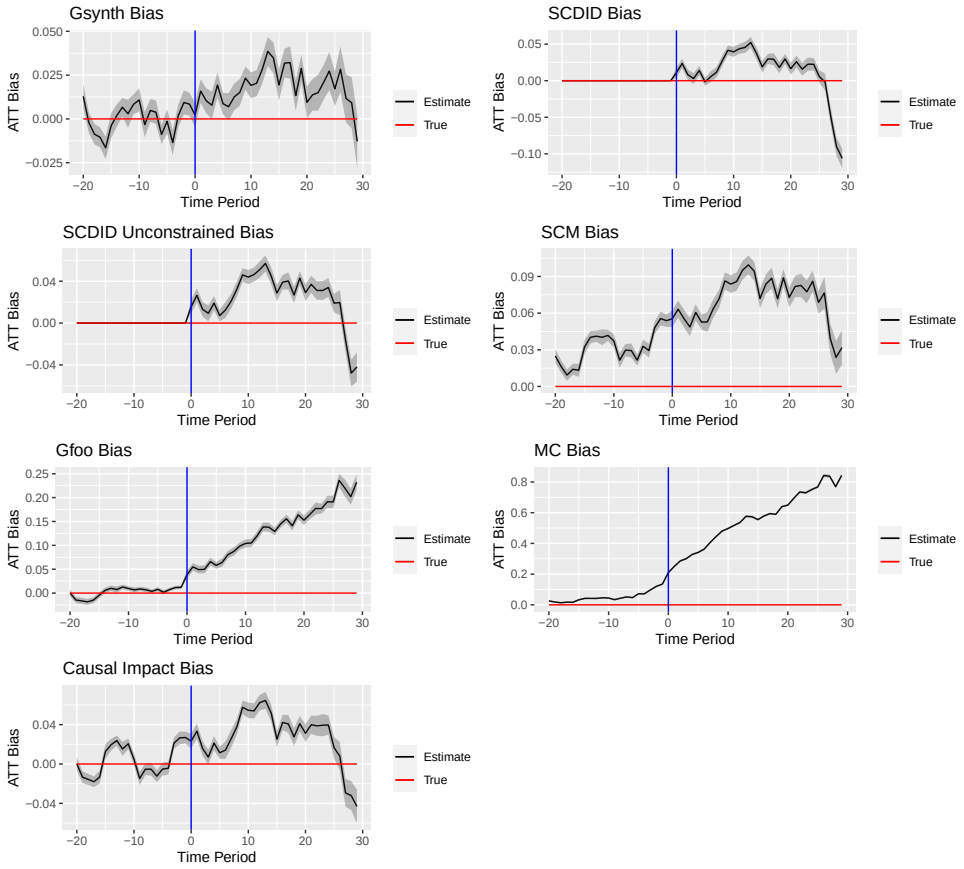
Xu, Yiqing. 2017. "Generalized Synthetic Control Method: Causal Inference with Interactive Fixed Effects Models." *Political Analysis* 25 (1): 57–76.

Appendix

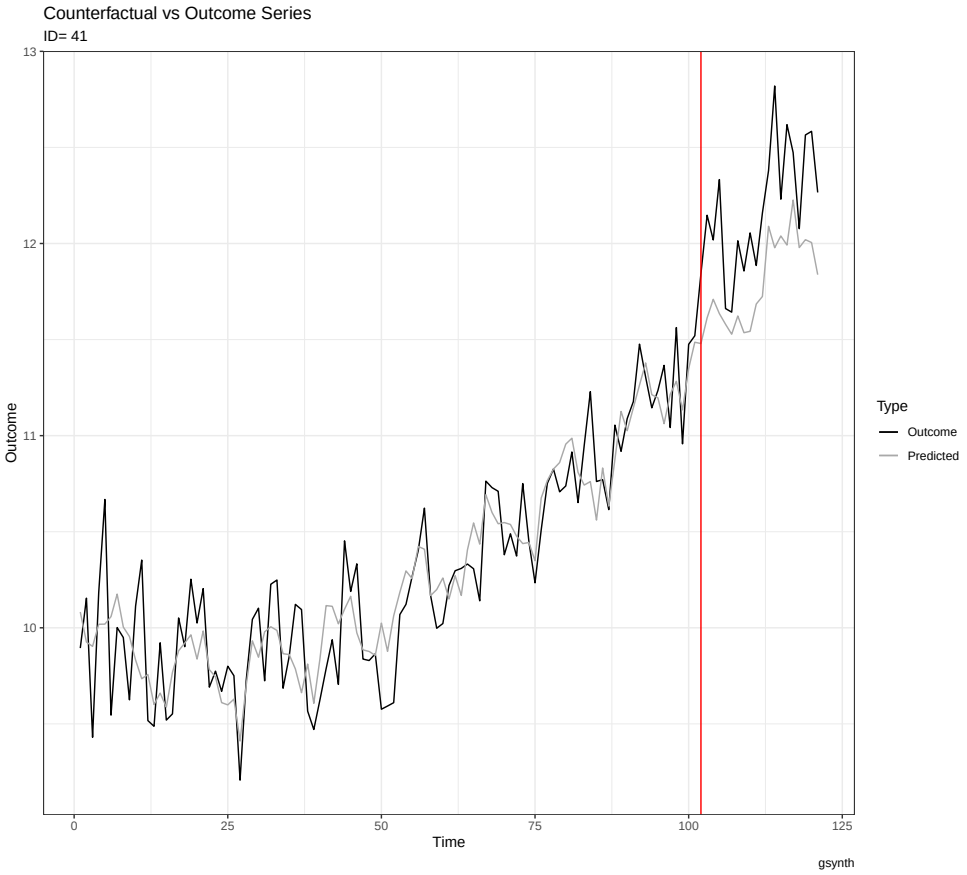
DGP Variations

For Loop Over DGPs

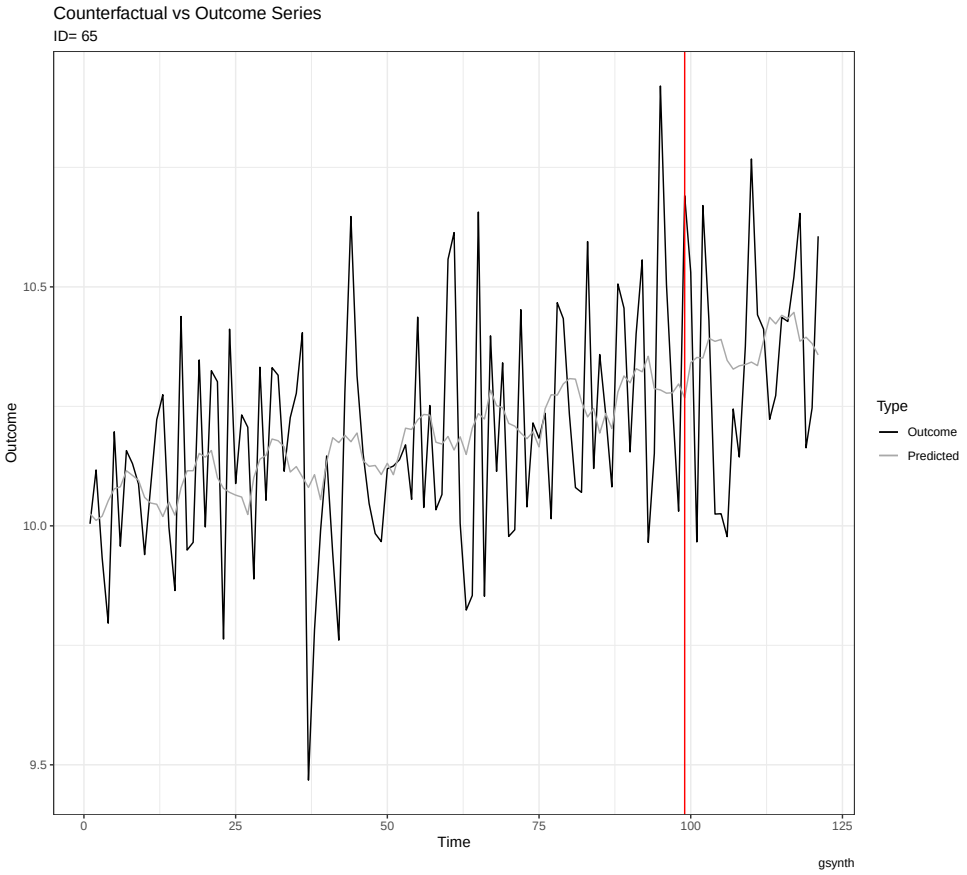
```
## [1] "aa_high_acf_loading_shift"
```



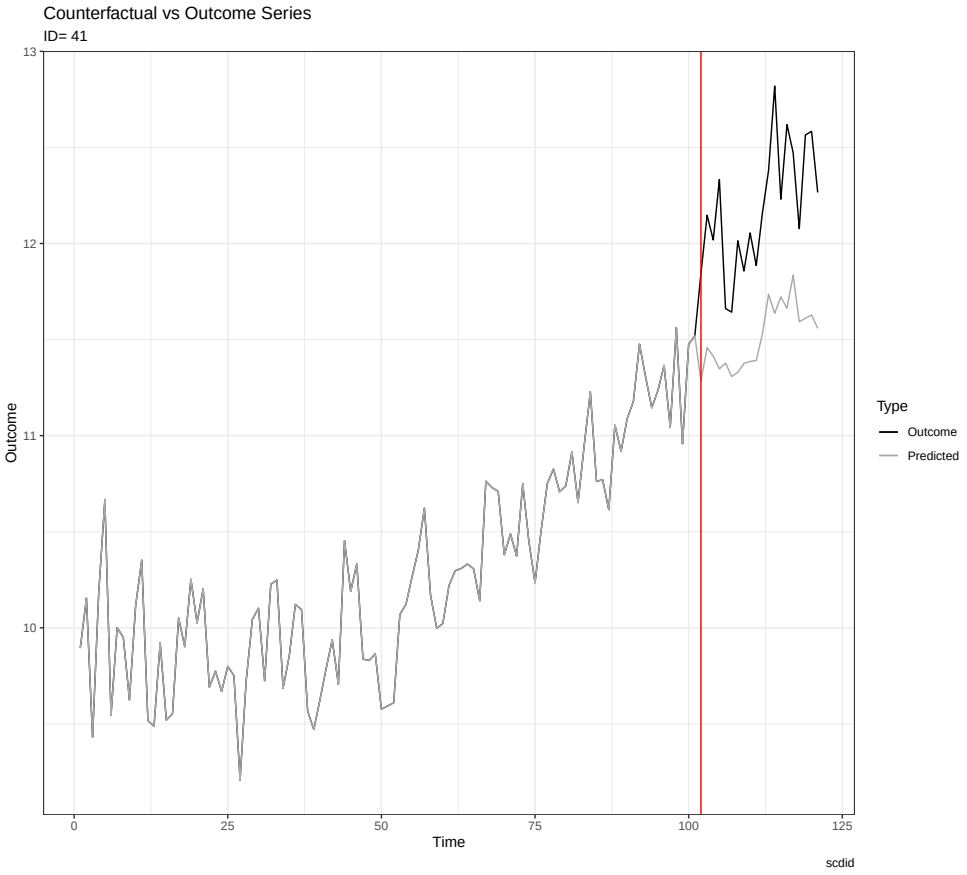

```
## [[1]]
```


Counterfactual vs Outcome Series
ID= 41
Outcome
Time
13
12
11
10
0
25
50
75
100
125
Type
Outcome
Predicted
gsynth

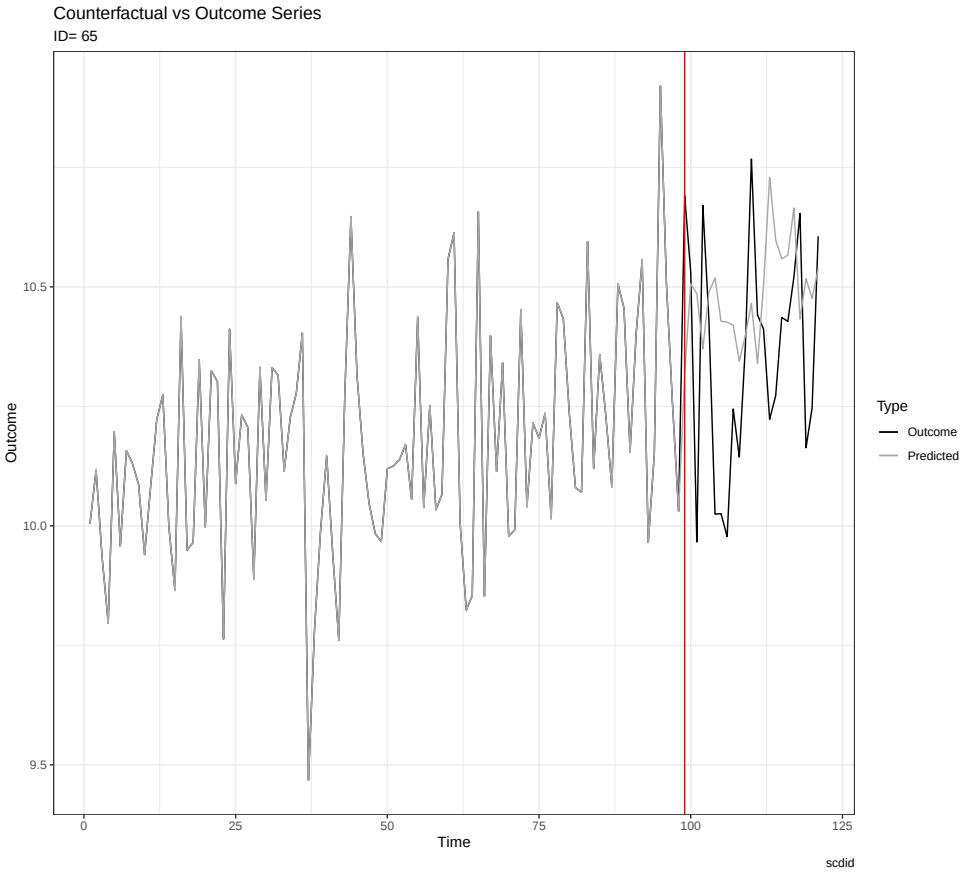
```
##
## [[2]]
```


Counterfactual vs Outcome Series
ID= 65
Outcome
10.5
10.0
9.5
0
25
50
75
100
125
Time
Type
Outcome
Predicted
gsynth

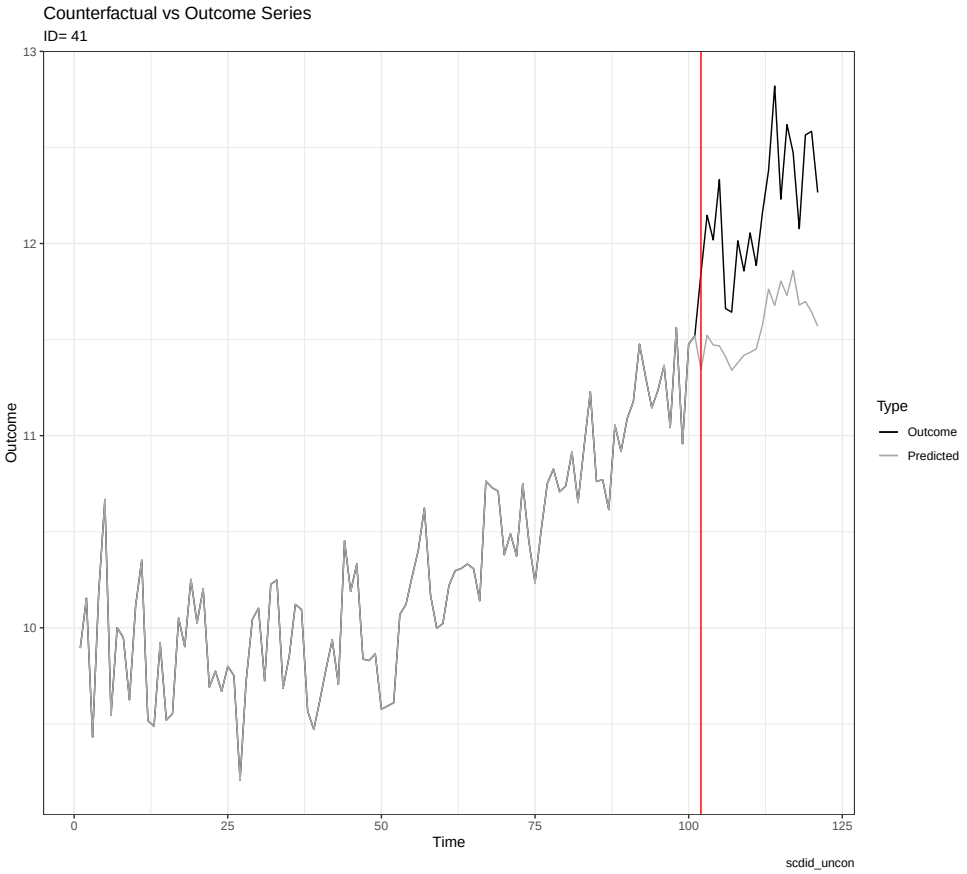
```
##
## [[1]]
```


Counterfactual vs Outcome Series
ID= 41
Outcome
Time
scdid
Type
Outcome
Predicted
13
12
11
10
0
25
50
75
100
125

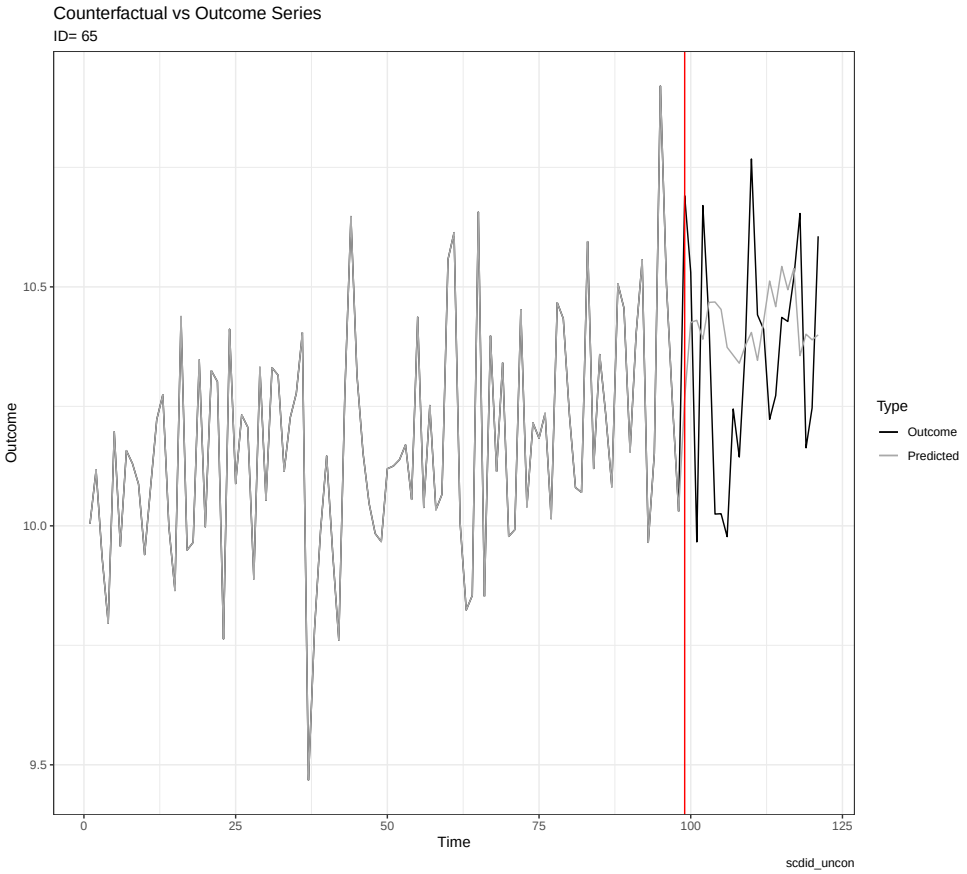
```
##
## [[2]]
```


Counterfactual vs Outcome Series
ID= 65
Outcome
10.5
10.0
9.5
0
25
50
75
100
125
Time
scdid
Type
Outcome
Predicted

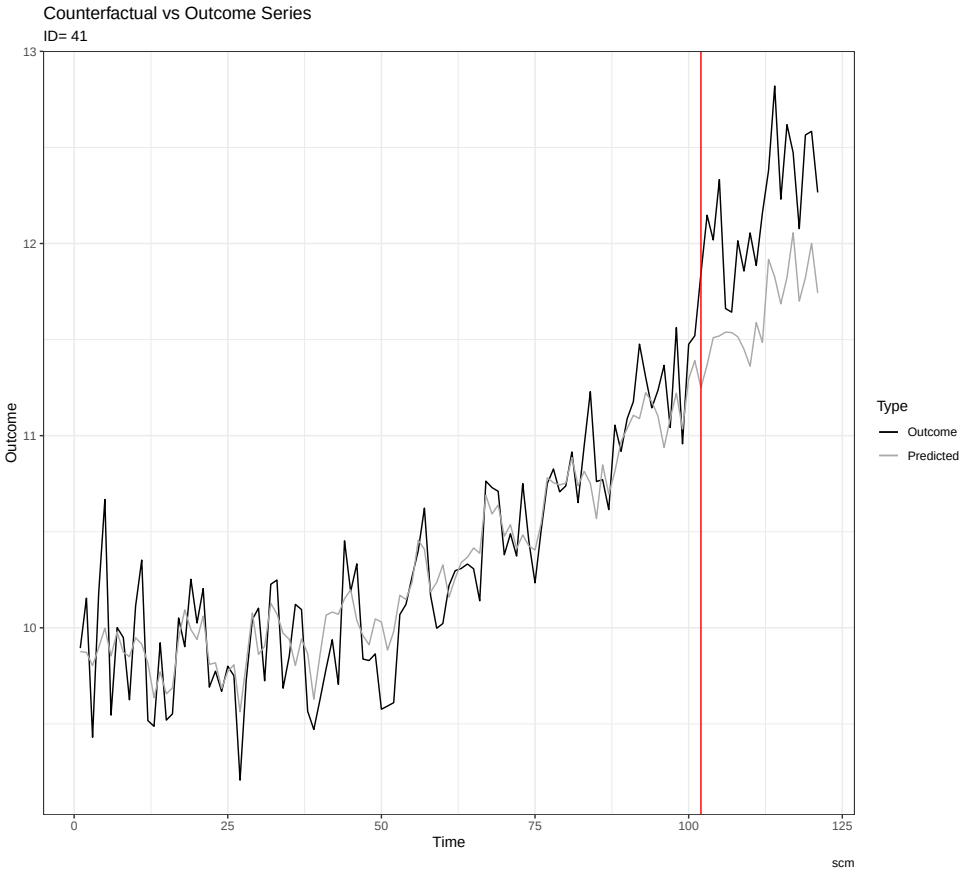
```
##
## [[1]]
```


Counterfactual vs Outcome Series
ID= 41
13
12
11
10
Outcome
0
25
50
75
100
125
Time
Type
Outcome
Predicted
scdid_uncon

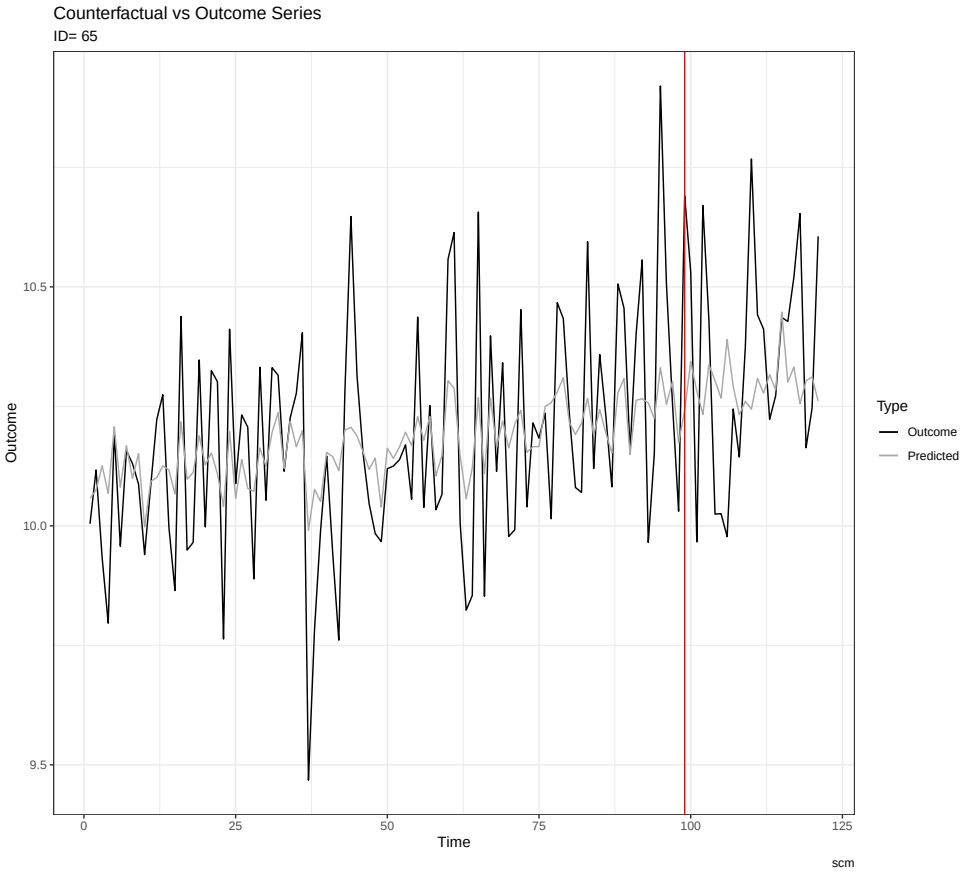
```
##
## [[2]]
```


Counterfactual vs Outcome Series
ID= 65
Outcome
10.5
10.0
9.5
0
25
50
75
100
125
Time
Type
Outcome
Predicted
scdid_uncon

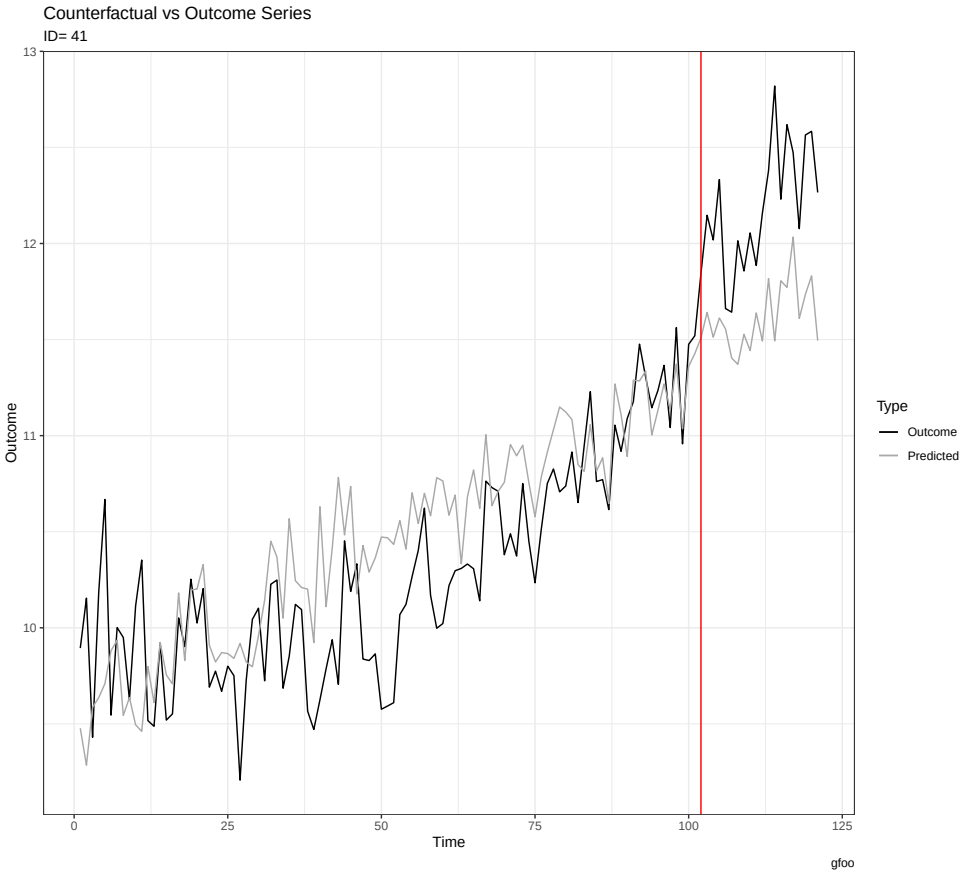
```
##
## [[1]]
```


Counterfactual vs Outcome Series
ID= 41
Outcome
Time
scm
Type
Outcome
Predicted
13
12
11
10
0
25
50
75
100
125

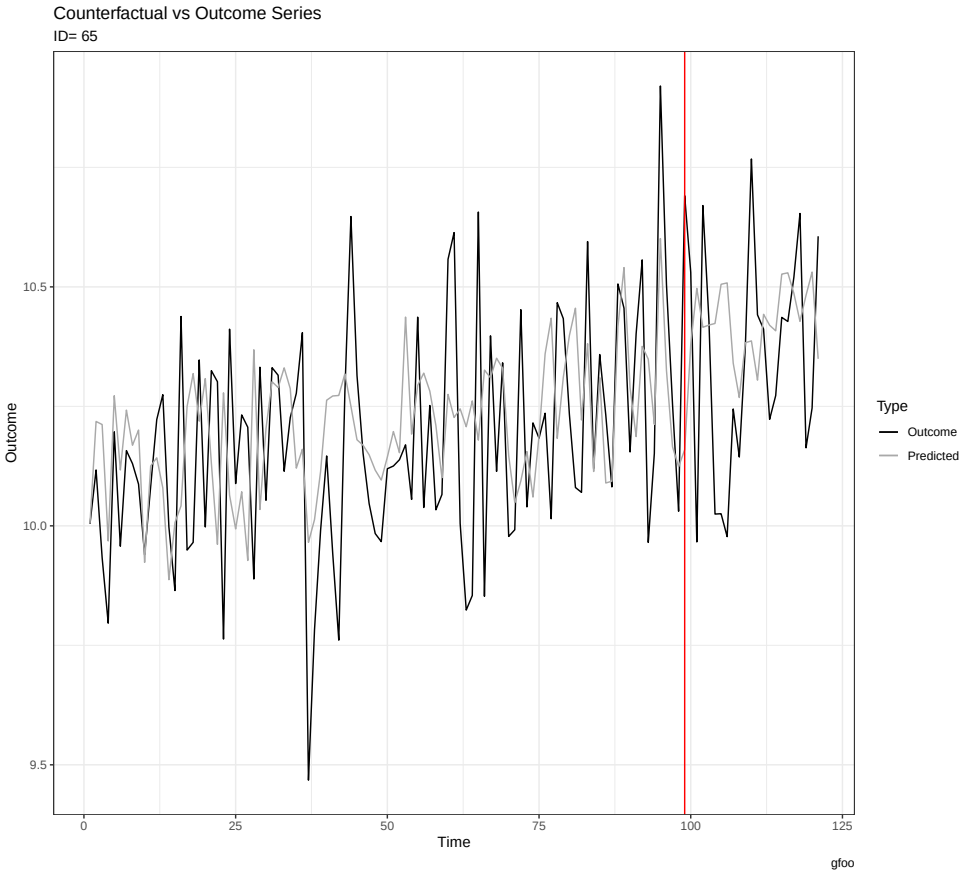
```
##
## [[2]]
```


Counterfactual vs Outcome Series
ID= 65
Outcome
10.5
10.0
9.5
0
25
50
75
100
125
Time
scm
Type
Outcome
Predicted

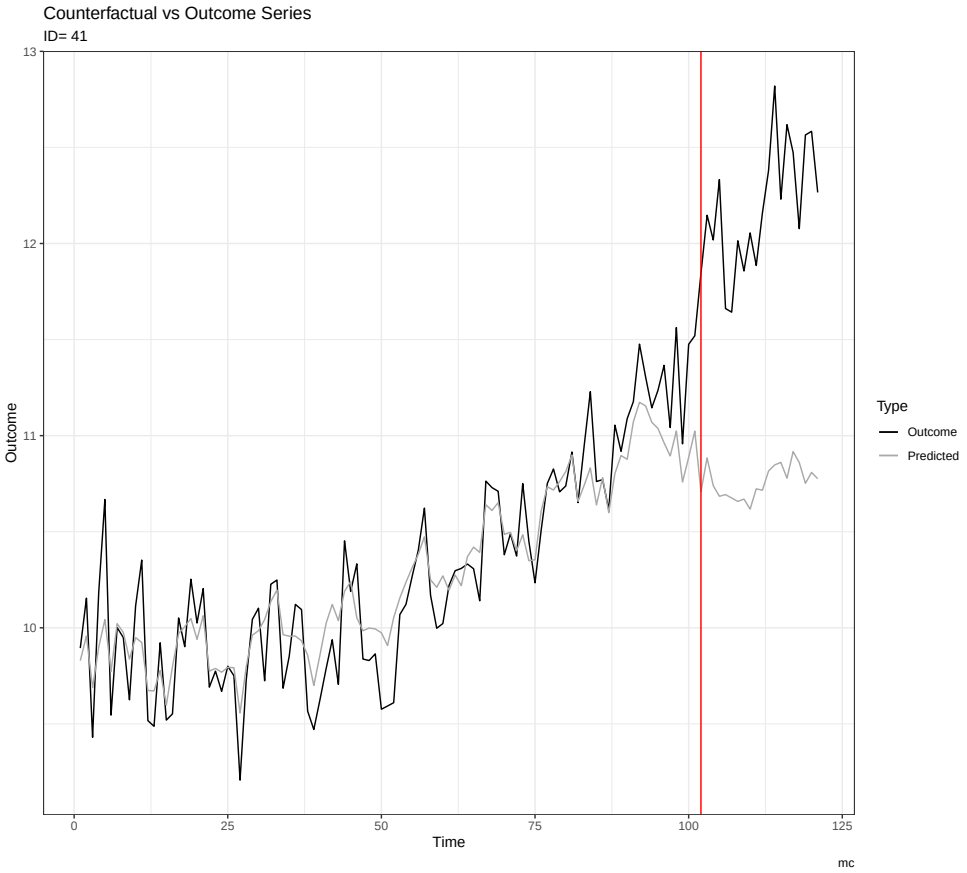
```
##
## [[1]]
```


Counterfactual vs Outcome Series
ID= 41
Outcome
Time
13
12
11
10
0
25
50
75
100
125
Type
Outcome
Predicted
gfoo

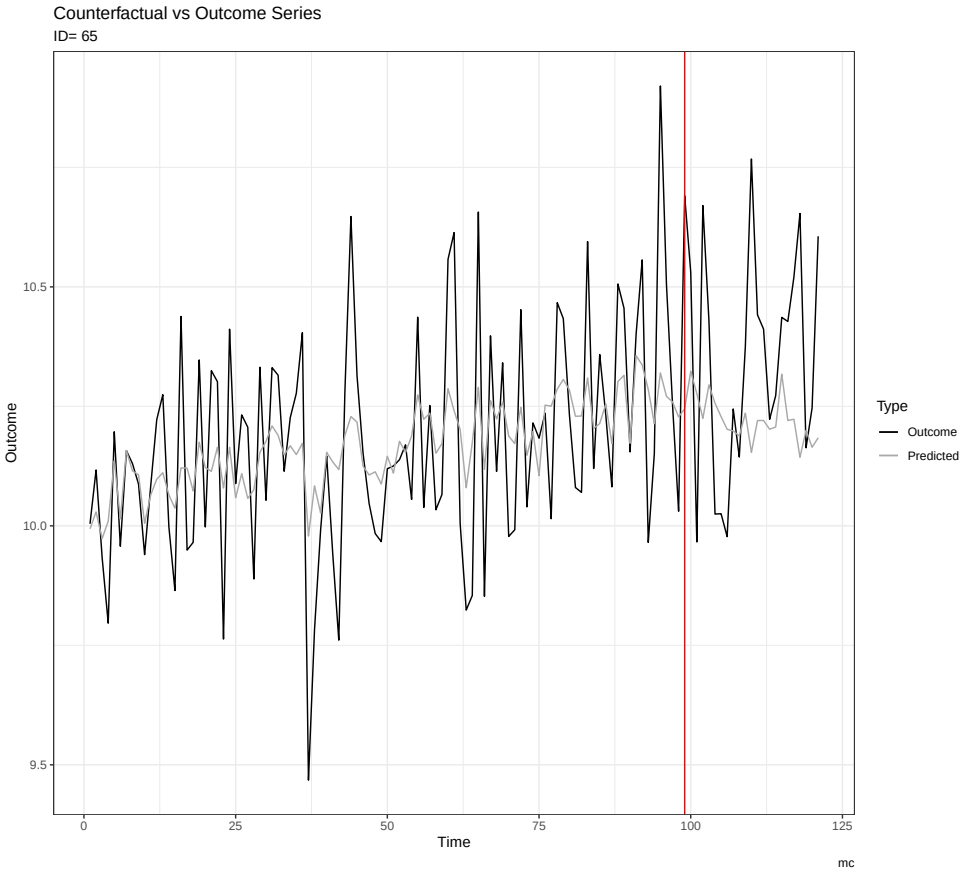
```
##
## [[2]]
```


Counterfactual vs Outcome Series
ID= 65
Outcome
10.5
10.0
9.5
0
25
50
75
100
125
Time
Type
Outcome
Predicted
gfoo

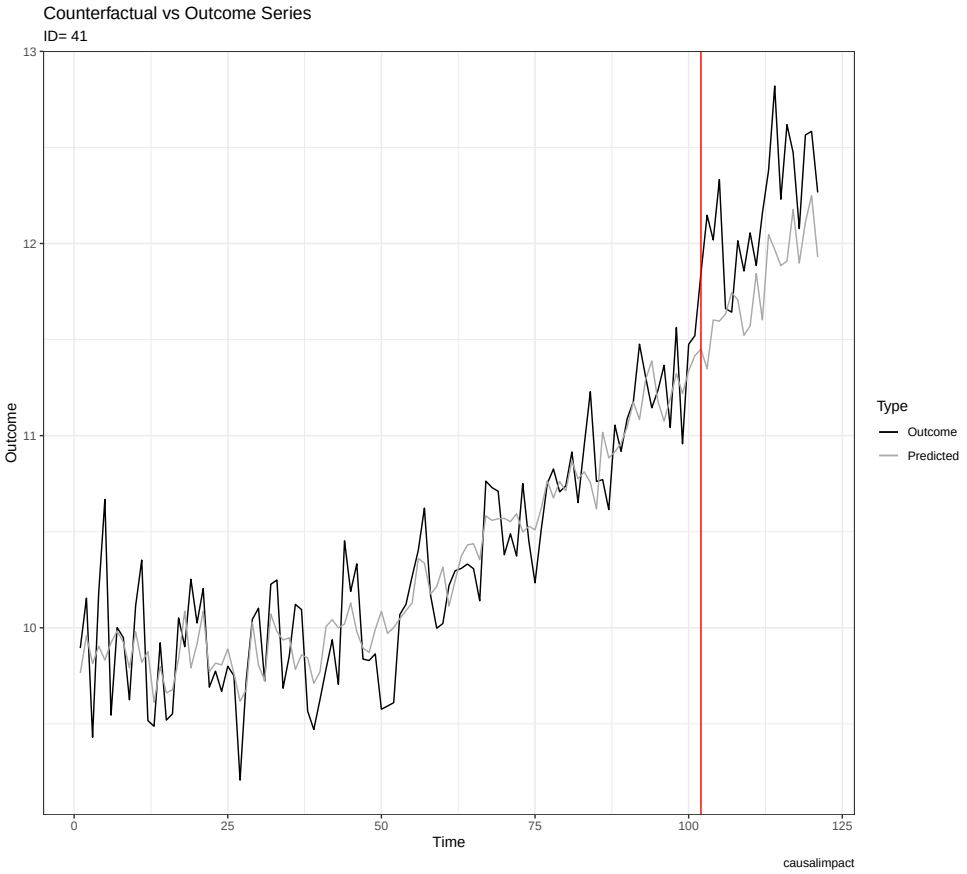
```
##
## [[1]]
```


Counterfactual vs Outcome Series
ID= 41
Outcome
Time
mc
Type
Outcome
Predicted
13
12
11
10
0
25
50
75
100
125

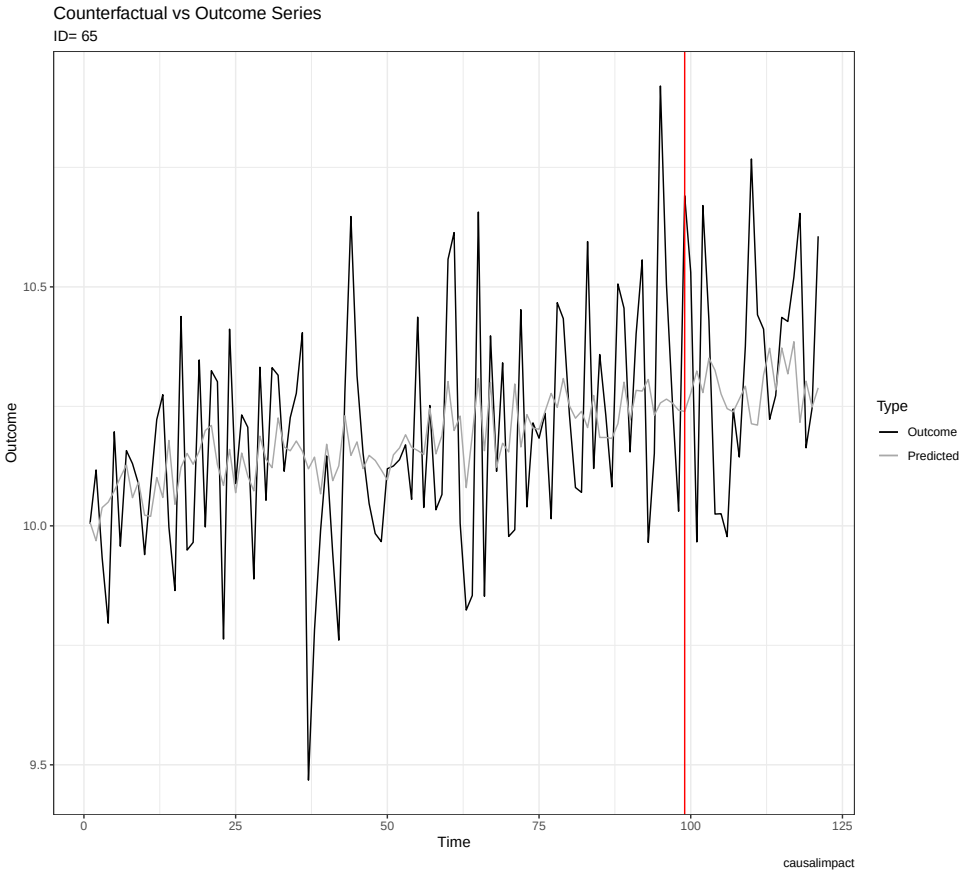
```
##
## [[2]]
```


Counterfactual vs Outcome Series
ID= 65
Outcome
10.5
10.0
9.5
0
25
50
75
100
125
Time
mc
Type
Outcome
Predicted

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

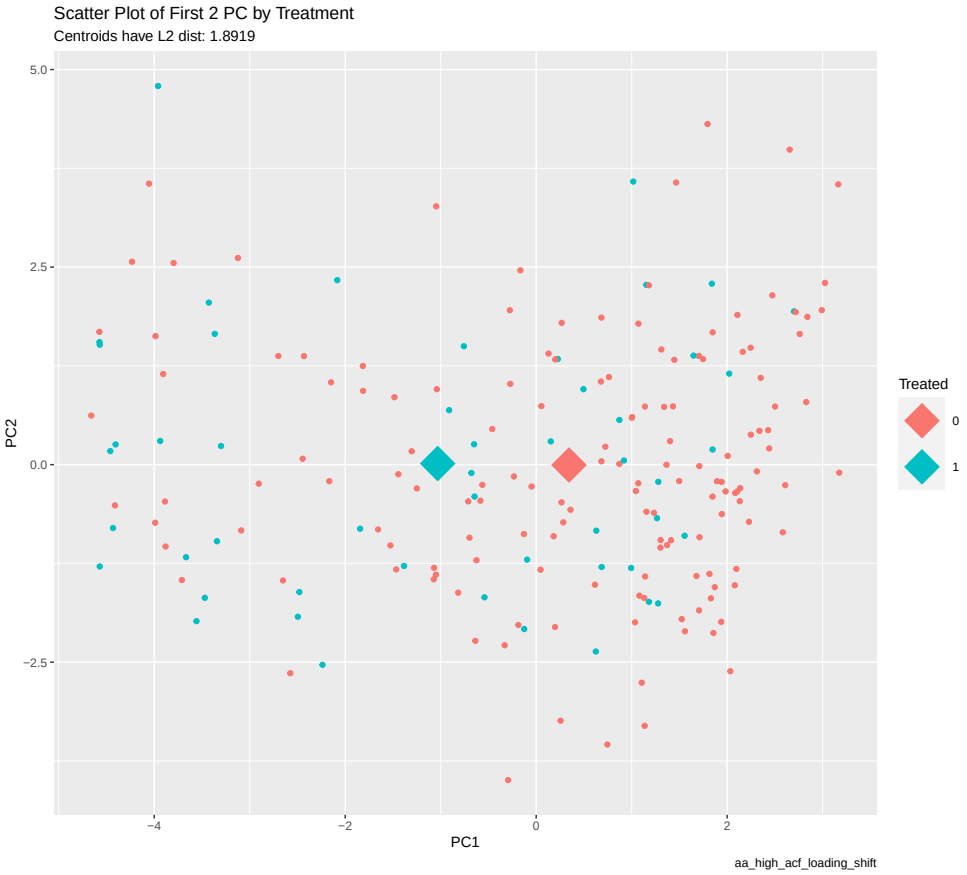

Counterfactual vs Outcome Series
ID= 41
Outcome
Time
13
12
11
10
0
25
50
75
100
125
Type
Outcome
Predicted
causalimpact

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




```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
## # A tibble: 9 x 8
##   vars          n1    n2 statistic    df        p    p.adj p.adj.signif
##   <chr>      <int> <int>     <dbl> <dbl>    <dbl>    <dbl> <chr>
## 1 curvature    150    50     0.250  86.0 0.803    0.803    ns
## 2 diff1_acf1   150    50    -0.631  81.5 0.53     0.681    ns
## 3 diff2_acf1   150    50     0.336  88.8 0.738    0.803    ns
## 4 e_acf1       150    50    -0.644  78.3 0.522    0.681    ns
## 5 entropy      150    50     3.51   70.1 0.000791 0.00142  **
## 6 linearity    150    50    -3.87   83.8 0.000212 0.000948 ***
## 7 spike        150    50     3.82   88.7 0.000245 0.000948 ***
## 8 trend        150    50    -3.78   74.0 0.000316 0.000948 ***
## 9 x_acf1       150    50    -3.54   74.6 0.000685 0.00142  **
```

Metrics by Method

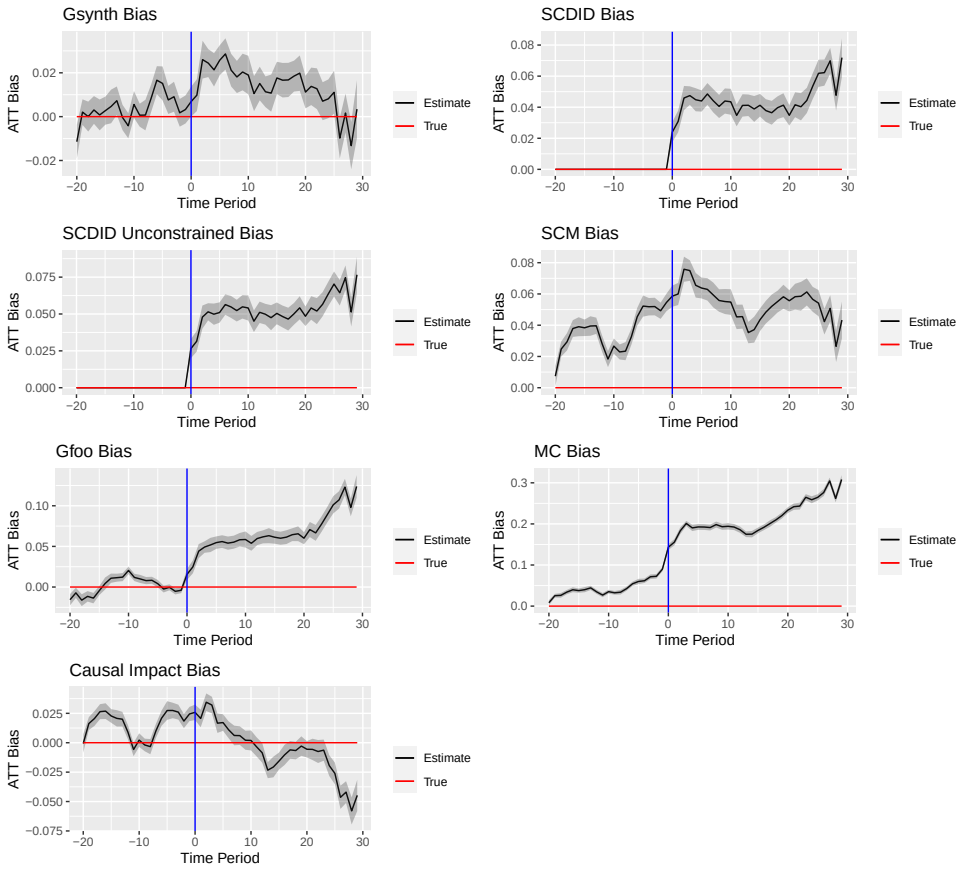
aa_high_acf_loading_shift

Method	gsynth	scdid	scdid_uncon	scm	gfoo	mc	causalimpact
coverage							

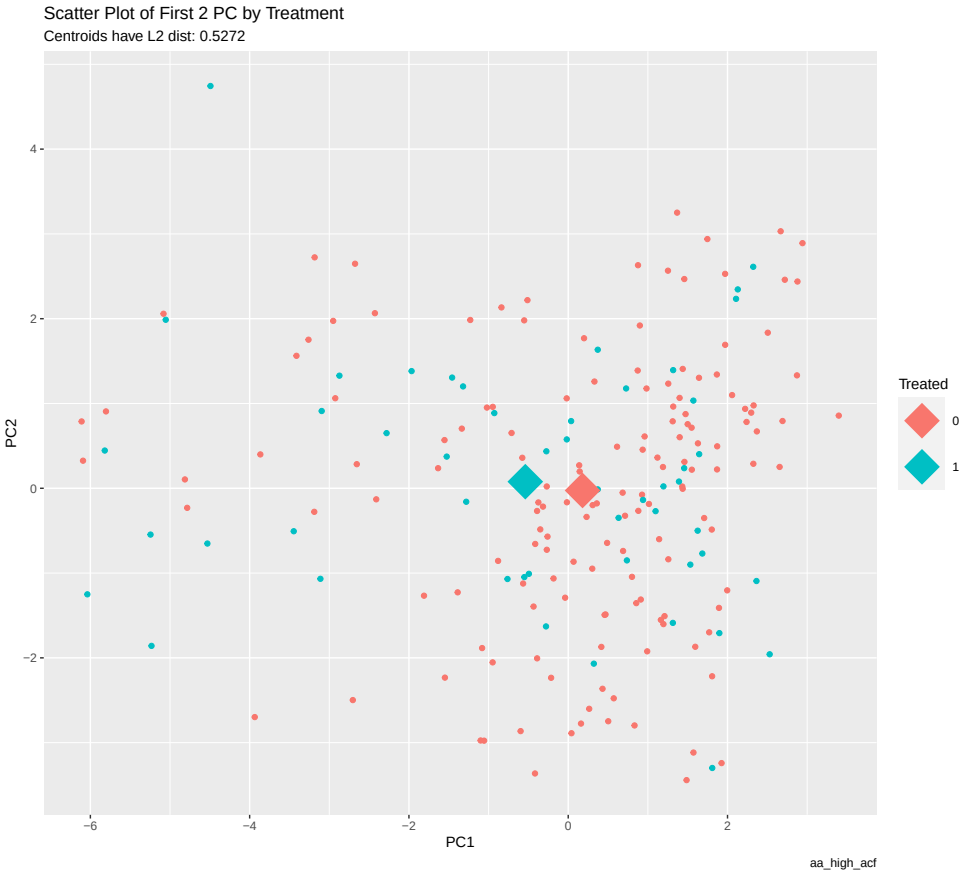
0	0.933	0.973	0.947	0.613	0.813	0.000	0.880
1	0.933	0.880	0.867	0.533	0.640	0.000	0.827
2	0.960	0.987	0.973	0.547	0.667	0.000	0.960
3	0.933	0.987	0.973	0.680	0.707	0.000	0.987
4	0.920	1.000	0.960	0.600	0.533	0.000	0.933
rmse							
0	0.215	0.223	0.217	0.230	0.241	0.416	0.236
1	0.223	0.236	0.230	0.244	0.255	0.462	0.247
2	0.224	0.237	0.231	0.242	0.260	0.526	0.247
3	0.226	0.237	0.230	0.239	0.258	0.540	0.245
4	0.223	0.241	0.234	0.242	0.267	0.562	0.248
bias							
0	0.002	0.011	0.016	0.055	0.038	0.210	0.023
1	0.016	0.024	0.026	0.063	0.055	0.251	0.033
2	0.010	0.008	0.013	0.056	0.049	0.286	0.016
3	0.008	0.003	0.009	0.049	0.050	0.301	0.007
4	0.019	0.014	0.019	0.061	0.066	0.328	0.021

Notes:

```
## [1] "aa_high_acf"
```




```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
## # A tibble: 9 x 8
##   vars          n1    n2 statistic    df      p p.adj p.adj.signif
##   <chr>      <int> <int>     <dbl> <dbl>  <dbl> <dbl> <chr>
## 1 curvature    150    50   -0.635   74.3 0.527  0.790 ns
## 2 diff1_acf1   150    50   -0.277   95.8 0.783  0.923 ns
## 3 diff2_acf1   150    50   -0.0973  86.1 0.923  0.923 ns
## 4 e_acf1       150    50    0.151   84.5 0.88   0.923 ns
## 5 entropy      150    50    2.03    66.0 0.0459 0.138 ns
## 6 linearity    150    50   -1.20    68.9 0.233  0.419 ns
## 7 spike        150    50    1.22    71.4 0.228  0.419 ns
## 8 trend        150    50   -2.19    71.0 0.0319 0.138 ns
## 9 x_acf1       150    50   -2.07    72.6 0.042  0.138 ns
```

Metrics by Method

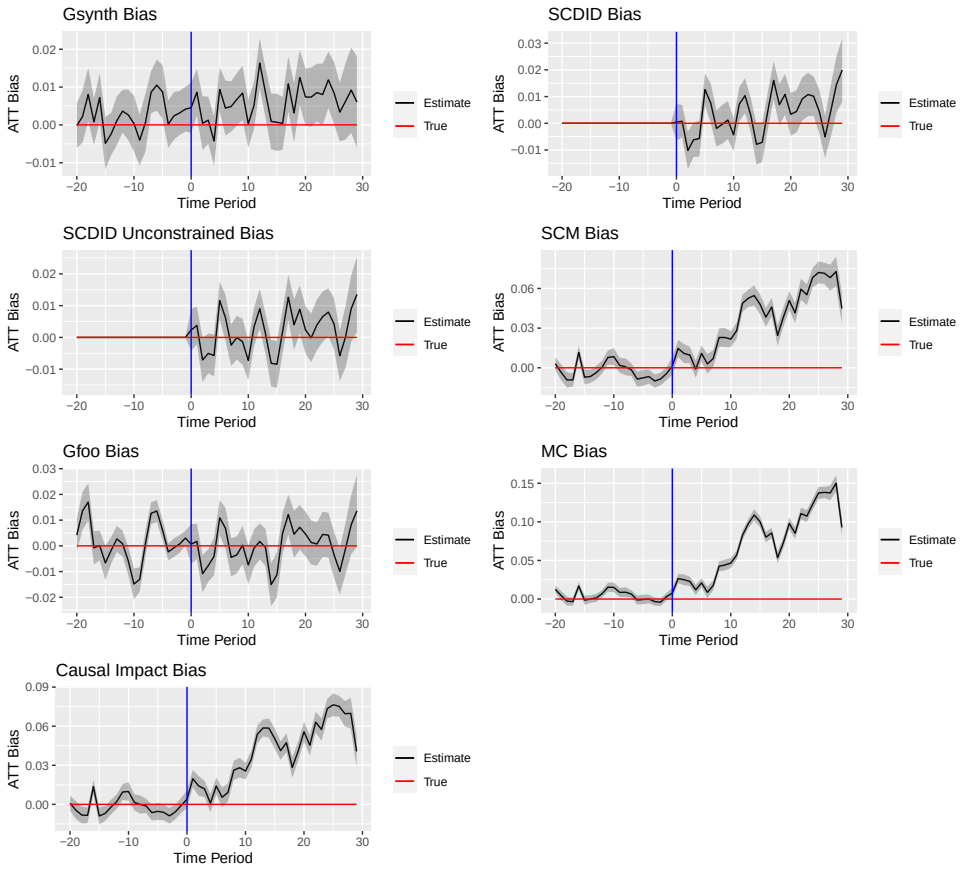
aa_high_acf

Method	gsynth	scdid	scdid_uncon	scm	gfoo	mc	causalimpact
coverage							

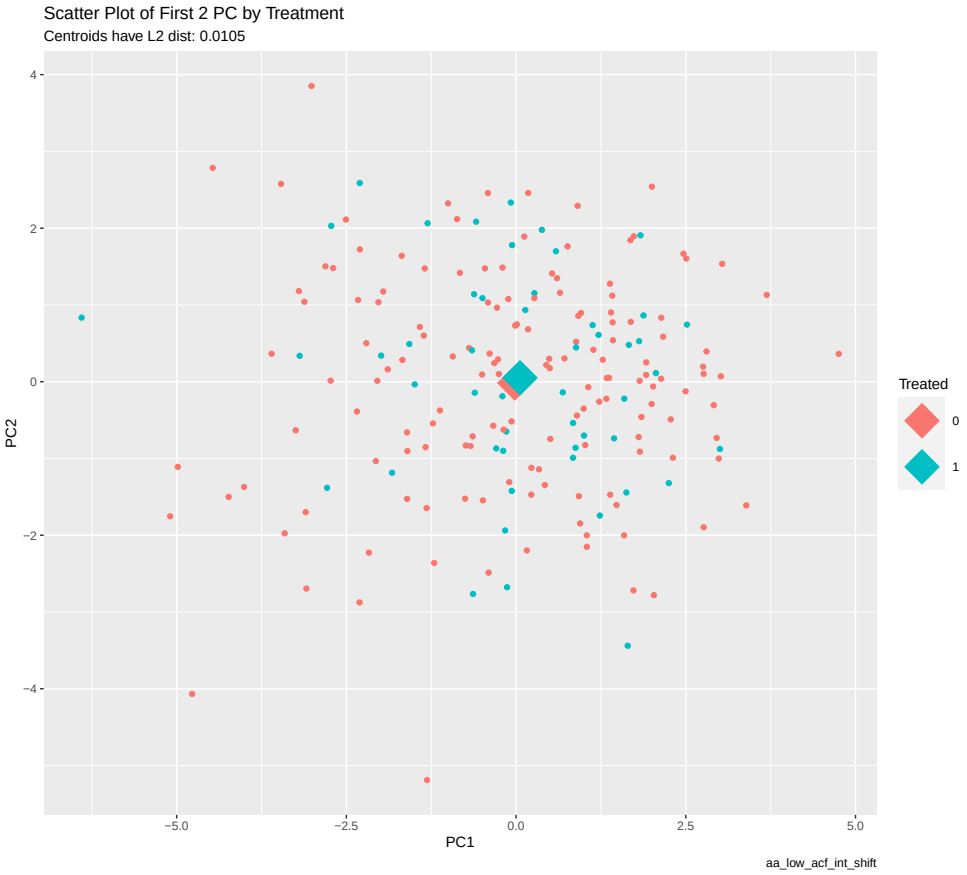
0	0.973	0.920	0.893	0.587	0.960	0.040	0.920
1	0.920	0.907	0.893	0.640	0.933	0.000	0.933
2	0.840	0.773	0.707	0.400	0.813	0.013	0.827
3	0.933	0.840	0.800	0.493	0.840	0.000	0.920
4	0.920	0.787	0.747	0.520	0.720	0.013	0.920
rmse							
0	0.221	0.238	0.236	0.239	0.245	0.329	0.237
1	0.228	0.250	0.247	0.250	0.257	0.359	0.244
2	0.228	0.250	0.246	0.249	0.260	0.381	0.240
3	0.238	0.260	0.255	0.261	0.272	0.419	0.251
4	0.233	0.256	0.252	0.253	0.269	0.414	0.242
bias							
0	0.007	0.024	0.027	0.059	0.016	0.143	0.026
1	0.010	0.031	0.031	0.060	0.025	0.156	0.021
2	0.026	0.046	0.048	0.076	0.044	0.183	0.034
3	0.024	0.047	0.051	0.075	0.049	0.201	0.032
4	0.021	0.045	0.050	0.066	0.052	0.190	0.017

Notes:

```
## [1] "aa_low_acf_int_shift"
```




```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
## # A tibble: 9 x 8
##   vars          n1    n2 statistic    df      p p.adj p.adj.signif
##   <chr>      <int> <int>     <dbl> <dbl>  <dbl> <dbl> <chr>
## 1 curvature    150    50    -0.967  83.1 0.336  0.605 ns
## 2 diff1_acf1   150    50    -0.333  84.3 0.74   0.812 ns
## 3 diff2_acf1   150    50    -1.49   82.6 0.14   0.42  ns
## 4 e_acf1       150    50     1.11   85.1 0.272  0.605 ns
## 5 entropy      150    50    -1.79   98.1 0.0765 0.42  ns
## 6 linearity    150    50     1.49   92.8 0.139  0.42  ns
## 7 spike        150    50     0.397 108.  0.692  0.812 ns
## 8 trend        150    50    -0.238  93.5 0.812  0.812 ns
## 9 x_acf1       150    50     0.607  92.3 0.545  0.812 ns
```

Metrics by Method

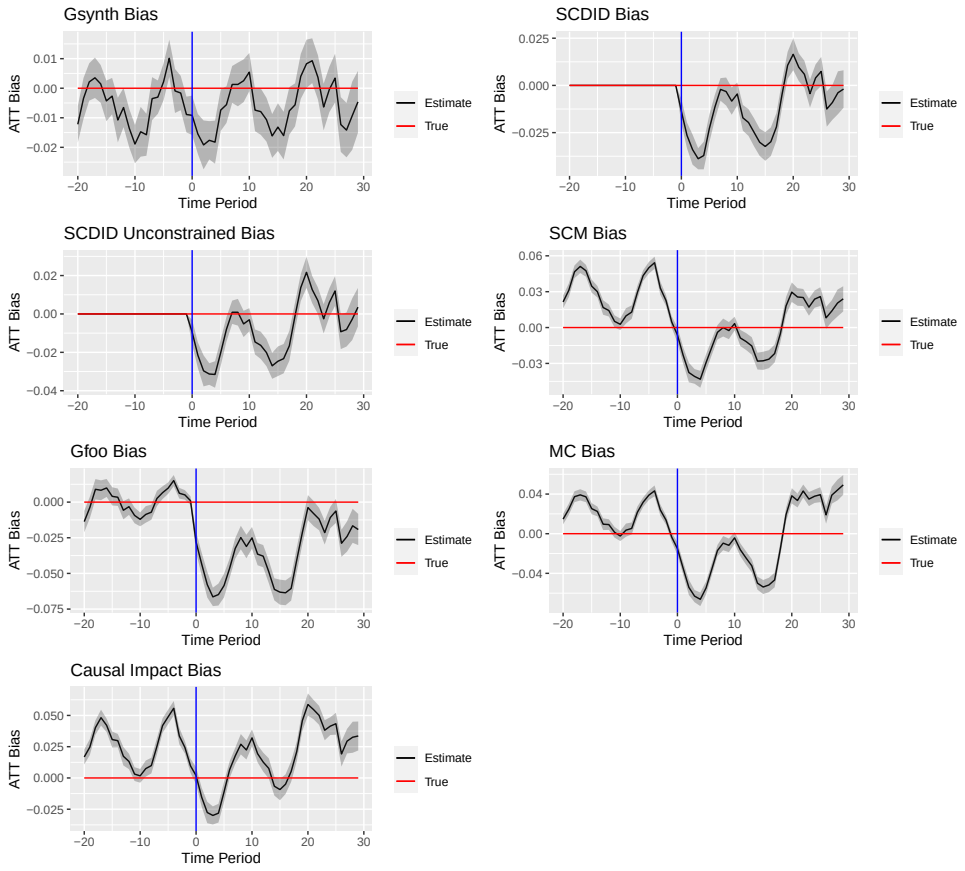
aa_low_acf_int_shift

Method	gsynth	scdid	scdid_uncon	scm	gfoo	mc	causalimpact
coverage							

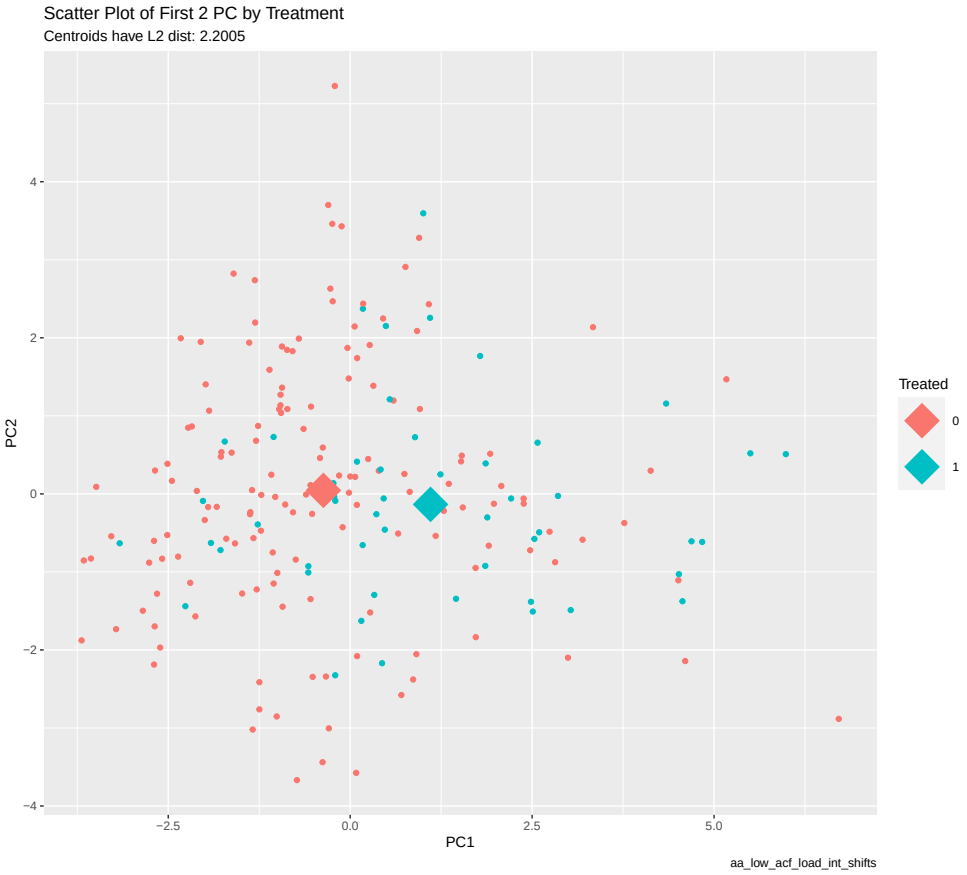
0	0.960	0.933	0.933	0.960	0.907	0.973	0.973
1	0.947	0.973	0.973	0.947	0.960	0.893	0.920
2	0.947	0.960	0.960	0.907	0.960	0.867	0.893
3	0.973	0.960	0.973	0.987	0.973	0.920	0.973
4	0.960	0.987	0.973	0.987	0.947	0.947	0.973
rmse							
0	0.209	0.212	0.211	0.216	0.227	0.214	0.222
1	0.211	0.213	0.211	0.215	0.230	0.216	0.221
2	0.209	0.211	0.210	0.217	0.229	0.217	0.224
3	0.210	0.215	0.213	0.219	0.230	0.221	0.227
4	0.209	0.214	0.212	0.218	0.231	0.220	0.225
bias							
0	0.005	0.000	0.002	0.001	0.001	0.008	0.004
1	0.009	0.001	0.004	0.015	0.002	0.026	0.020
2	0.000	−0.010	−0.007	0.011	−0.011	0.025	0.014
3	0.001	−0.006	−0.005	0.010	−0.008	0.023	0.012
4	−0.004	−0.006	−0.006	−0.001	−0.004	0.012	0.001

Notes:

```
## [1] "aa_low_acf_load_int_shifts"
```




```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
## # A tibble: 9 x 8
##   vars          n1    n2 statistic    df          p     p.adj p.adj.signif
##   <chr>      <int> <int>     <dbl> <dbl>      <dbl>     <dbl> <chr>
## 1 curvature    150    50     1.18   97.6 0.239      0.307     ns
## 2 diff1_acf1   150    50    -2.55   87.3 0.0127     0.0229    *
## 3 diff2_acf1   150    50    -0.870  88.9 0.387      0.387     ns
## 4 e_acf1       150    50    -2.07  104.  0.0407     0.0610    ns
## 5 entropy      150    50     3.05   77.4 0.00311    0.00700   **
## 6 linearity    150    50    -0.951 109.  0.344      0.387     ns
## 7 spike        150    50     4.74   95.6 0.0000074  0.0000222 ****
## 8 trend        150    50    -4.93   77.1 0.00000452 0.0000222 ****
## 9 x_acf1       150    50    -4.86   78.5 0.0000058  0.0000222 ****
```

Metrics by Method

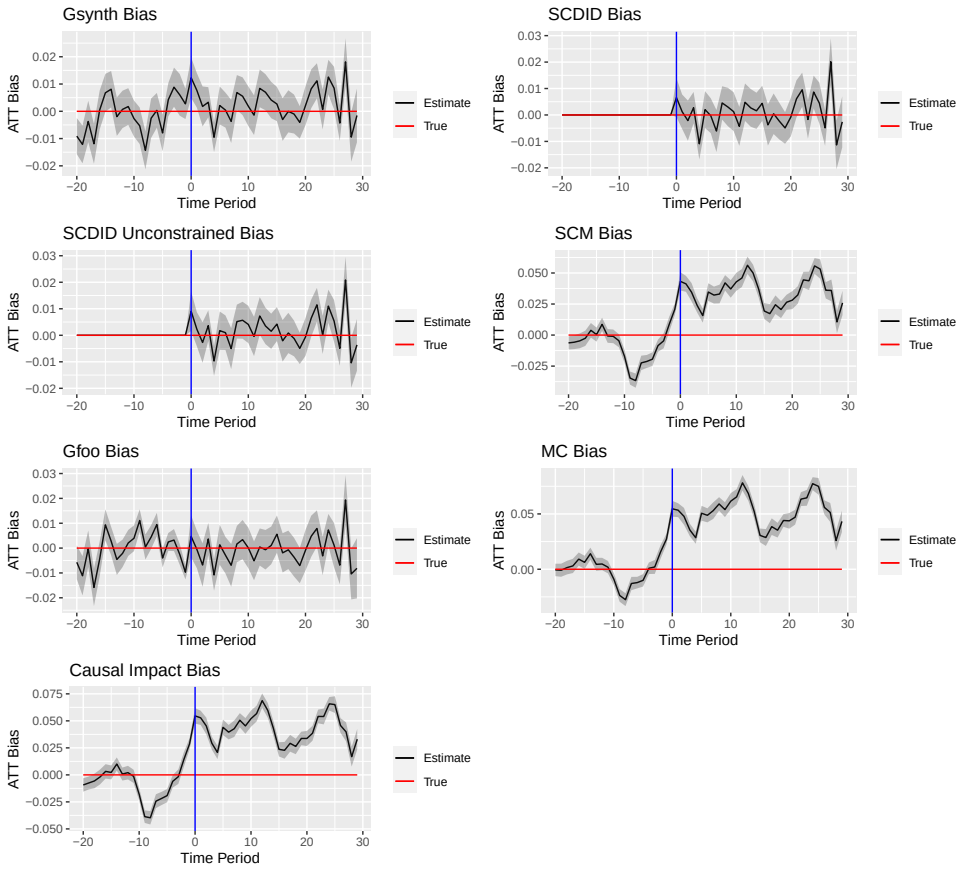
aa_low_acf_load_int_shifts

Method	gsynth	scdid	scdid_uncon	scm	gfoo	mc	causalimpact
coverage							

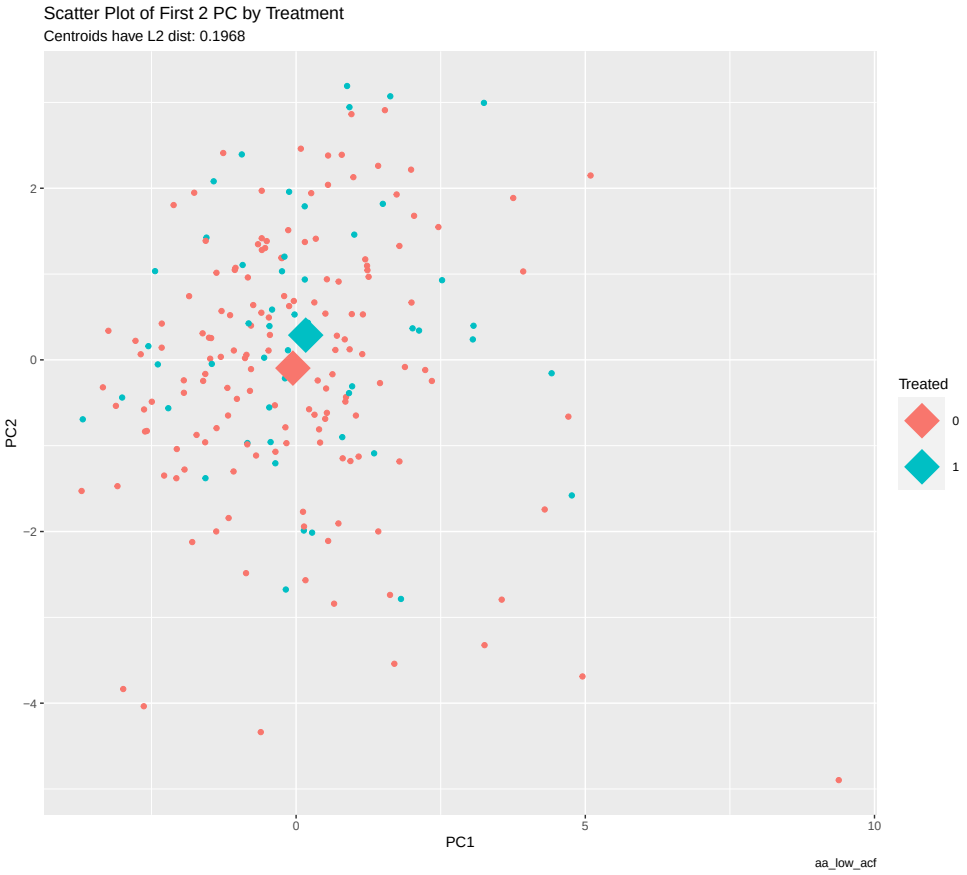
0	0.933	0.960	0.947	0.933	0.907	0.960	0.973
1	0.920	0.840	0.907	0.907	0.680	0.787	0.933
2	0.880	0.787	0.827	0.747	0.693	0.613	0.840
3	0.880	0.800	0.840	0.787	0.600	0.480	0.827
4	0.880	0.800	0.840	0.747	0.627	0.427	0.867
rmse							
0	0.212	0.216	0.214	0.217	0.243	0.216	0.228
1	0.214	0.219	0.215	0.220	0.250	0.219	0.231
2	0.217	0.225	0.220	0.225	0.260	0.227	0.235
3	0.212	0.221	0.216	0.220	0.268	0.225	0.231
4	0.213	0.223	0.219	0.222	0.273	0.229	0.230
bias							
0	−0.009	−0.014	−0.009	−0.006	−0.028	−0.015	0.002
1	−0.015	−0.026	−0.021	−0.023	−0.043	−0.035	−0.015
2	−0.019	−0.034	−0.030	−0.038	−0.058	−0.054	−0.028
3	−0.018	−0.039	−0.031	−0.041	−0.066	−0.063	−0.030
4	−0.018	−0.037	−0.032	−0.043	−0.065	−0.066	−0.028

Notes:

```
## [1] "aa_low_acf"
```




```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
## # A tibble: 9 x 8
##   vars          n1    n2 statistic    df      p p.adj p.adj.signif
##   <chr>      <int> <int>     <dbl> <dbl>  <dbl> <dbl> <chr>
## 1 curvature    150    50     0.748  90.0 0.457  0.686 ns
## 2 diff1_acf1   150    50    -1.33   82.7 0.188  0.686 ns
## 3 diff2_acf1   150    50    -0.979  81.1 0.33   0.686 ns
## 4 e_acf1       150    50    -1.92   91.7 0.0573 0.516 ns
## 5 entropy      150    50     0.356  87.5 0.722  0.722 ns
## 6 linearity    150    50     0.483  83.3 0.63   0.709 ns
## 7 spike        150    50    -0.876  74.9 0.384  0.686 ns
## 8 trend        150    50     0.510 106.  0.611  0.709 ns
## 9 x_acf1       150    50    -0.921 105.  0.359  0.686 ns
```

Metrics by Method

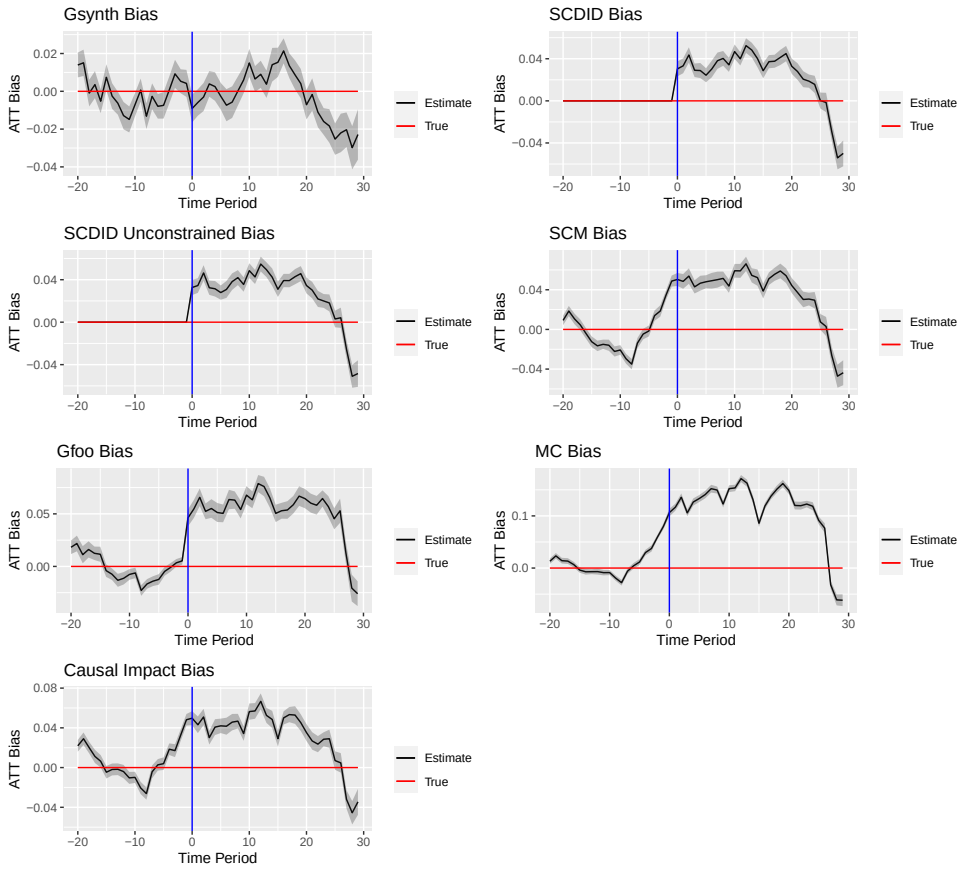
aa_low_acf

Method	gsynth	scdid	scdid_uncon	scm	gfoo	mc	causalimpact
coverage							

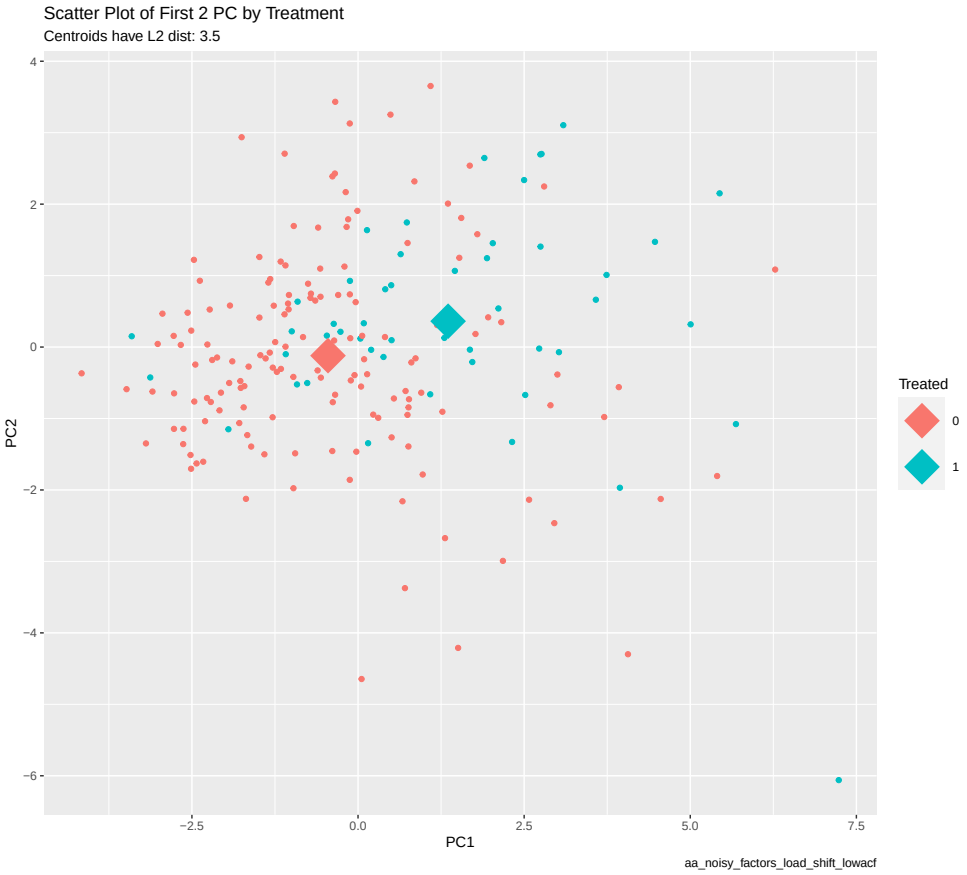
0	0.947	0.947	0.947	0.733	0.907	0.587	0.653
1	0.960	0.947	0.960	0.707	0.960	0.560	0.613
2	0.947	0.920	0.920	0.773	0.907	0.600	0.707
3	0.987	0.987	0.987	0.920	0.987	0.840	0.893
4	0.973	0.960	0.960	0.973	0.973	0.947	0.973
rmse							
0	0.210	0.211	0.209	0.218	0.229	0.218	0.226
1	0.209	0.210	0.208	0.217	0.233	0.216	0.224
2	0.206	0.207	0.205	0.213	0.227	0.214	0.219
3	0.208	0.211	0.209	0.213	0.230	0.212	0.218
4	0.206	0.209	0.206	0.210	0.228	0.208	0.215
bias							
0	0.012	0.007	0.009	0.043	0.005	0.055	0.055
1	0.008	0.001	0.002	0.041	−0.001	0.053	0.053
2	0.002	−0.002	−0.003	0.035	−0.007	0.048	0.045
3	0.003	0.003	0.004	0.024	0.004	0.036	0.029
4	−0.010	−0.011	−0.010	0.016	−0.011	0.029	0.021

Notes:

```
## [1] "aa_noisy_factors_load_shift_lowacf"
```




```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
## # A tibble: 9 x 8
##   vars          n1    n2 statistic    df          p      p.adj p.adj.signif
##   <chr>      <int> <int>     <dbl> <dbl>      <dbl>      <dbl> <chr>
## 1 curvature    150    50    -0.978  75.7 0.331      0.331      ns
## 2 diff1_acf1   150    50    -5.18   78.1 0.00000169 0.00000507 ****
## 3 diff2_acf1   150    50    -3.54   86.8 0.000654   0.00118    **
## 4 e_acf1       150    50    -5.24   74.7 0.00000144 0.00000507 ****
## 5 entropy      150    50     4.19   69.0 0.00008    0.00018    ***
## 6 linearity    150    50    -2.00   83.4 0.0489     0.0550     ns
## 7 spike        150    50     2.67   90.9 0.00903    0.0116     *
## 8 trend        150    50    -3.14   77.0 0.00237    0.00356    **
## 9 x_acf1       150    50    -5.23   78.8 0.00000137 0.00000507 ****
```

Metrics by Method

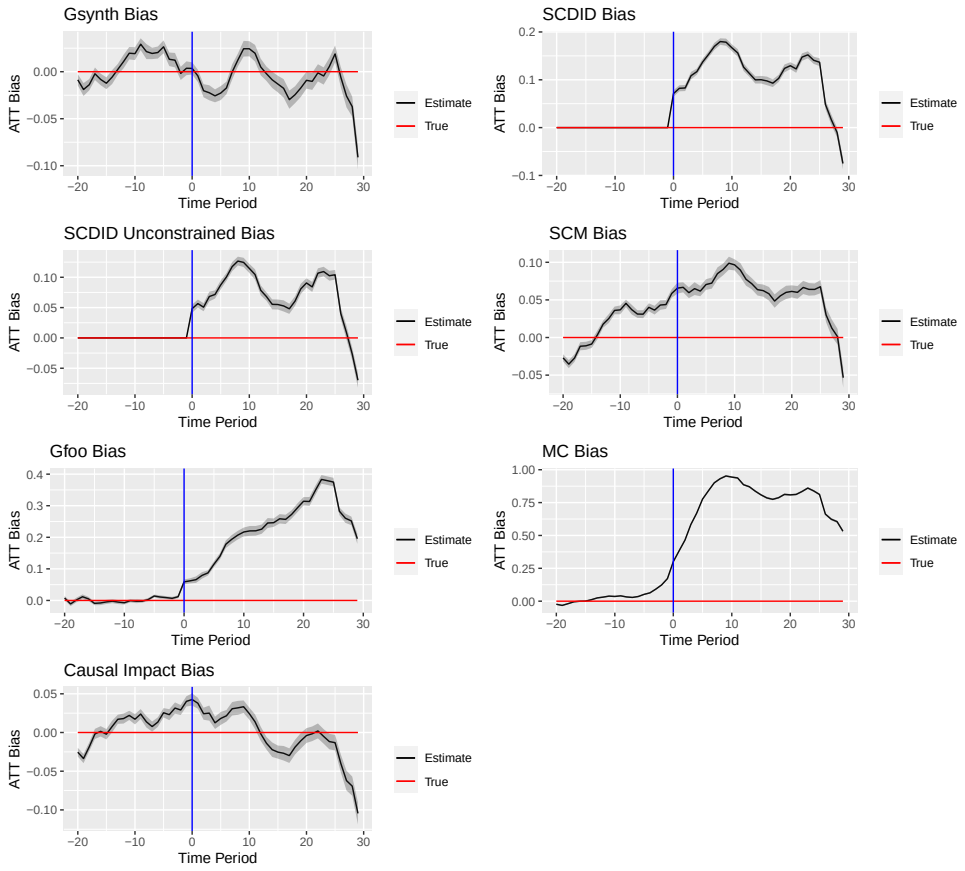
aa_noisy_factors_load_shift_lowacf

Method	gsynth	scdid	scdid_uncon	scm	gfoo	mc	causalimpact
coverage							

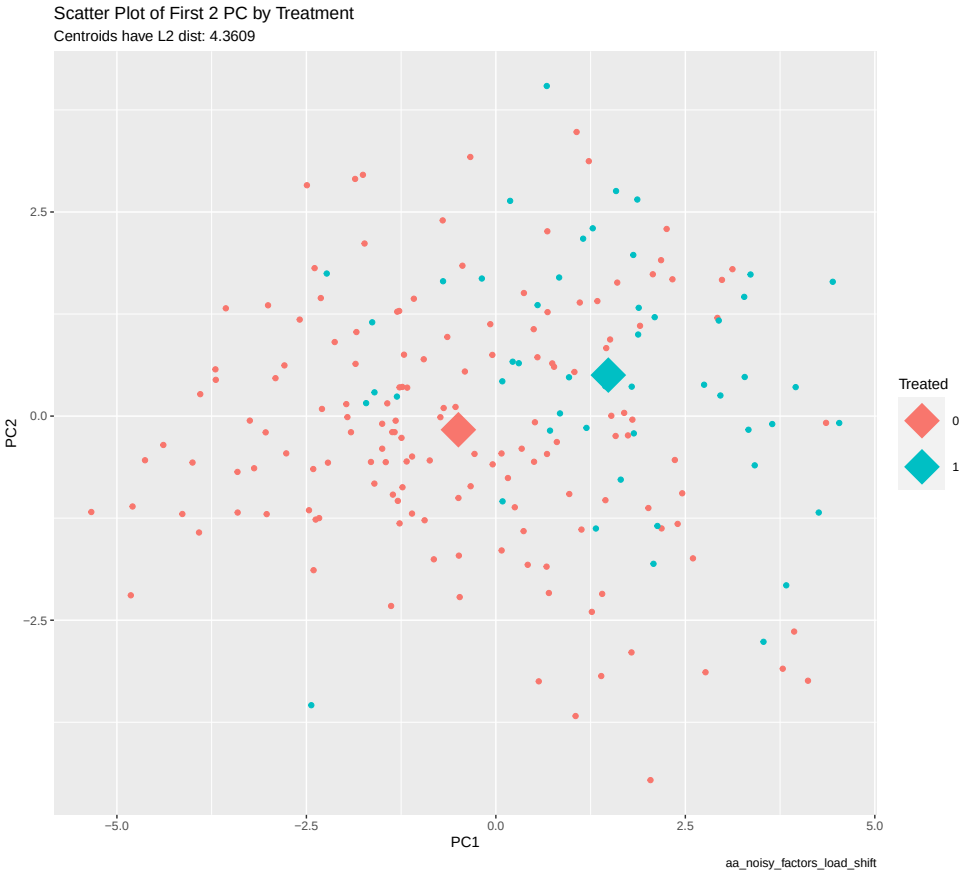
0	0.933	0.933	0.880	0.680	0.800	0.107	0.693
1	0.933	0.840	0.827	0.667	0.680	0.027	0.760
2	0.933	0.813	0.800	0.720	0.760	0.027	0.787
3	0.987	0.960	0.880	0.787	0.800	0.107	0.947
4	0.933	0.893	0.853	0.733	0.680	0.027	0.813
rmse							
0	0.223	0.230	0.228	0.235	0.261	0.252	0.247
1	0.227	0.237	0.234	0.241	0.271	0.268	0.254
2	0.231	0.249	0.248	0.250	0.287	0.292	0.264
3	0.239	0.252	0.250	0.251	0.277	0.274	0.262
4	0.238	0.245	0.243	0.247	0.261	0.271	0.256
bias							
0	−0.009	0.030	0.033	0.050	0.046	0.106	0.050
1	−0.006	0.033	0.035	0.048	0.055	0.117	0.043
2	−0.003	0.044	0.046	0.054	0.066	0.136	0.051
3	0.004	0.029	0.032	0.043	0.052	0.106	0.030
4	0.002	0.029	0.031	0.047	0.055	0.127	0.041

Notes:

```
## [1] "aa_noisy_factors_load_shift"
```




```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
## # A tibble: 9 x 8
##   vars          n1    n2 statistic    df        p         p.adj p.adj.signif
##   <chr>      <int> <int>     <dbl> <dbl>    <dbl>         <dbl> <chr>
## 1 curvature    150    50   -1.98    72.5 5.14e- 2 0.0578        ns
## 2 diff1_acf1   150    50   -6.37    80.1 1.13e- 8 0.0000000254  ****
## 3 diff2_acf1   150    50    0.0541  94.0 9.57e- 1 0.957         ns
## 4 e_acf1       150    50   -7.03    78.1 7.00e-10 0.00000000315 ****
## 5 entropy      150    50    4.21    80.0 6.63e- 5 0.0000994     ****
## 6 linearity    150    50   -3.10    92.6 2.60e- 3 0.00334       **
## 7 spike        150    50    6.00   178.  1.07e- 8 0.0000000254  ****
## 8 trend        150    50   -5.97   111.  2.86e- 8 0.0000000515  ****
## 9 x_acf1       150    50   -6.88   117.  3.21e-10 0.00000000289 ****
```

Metrics by Method

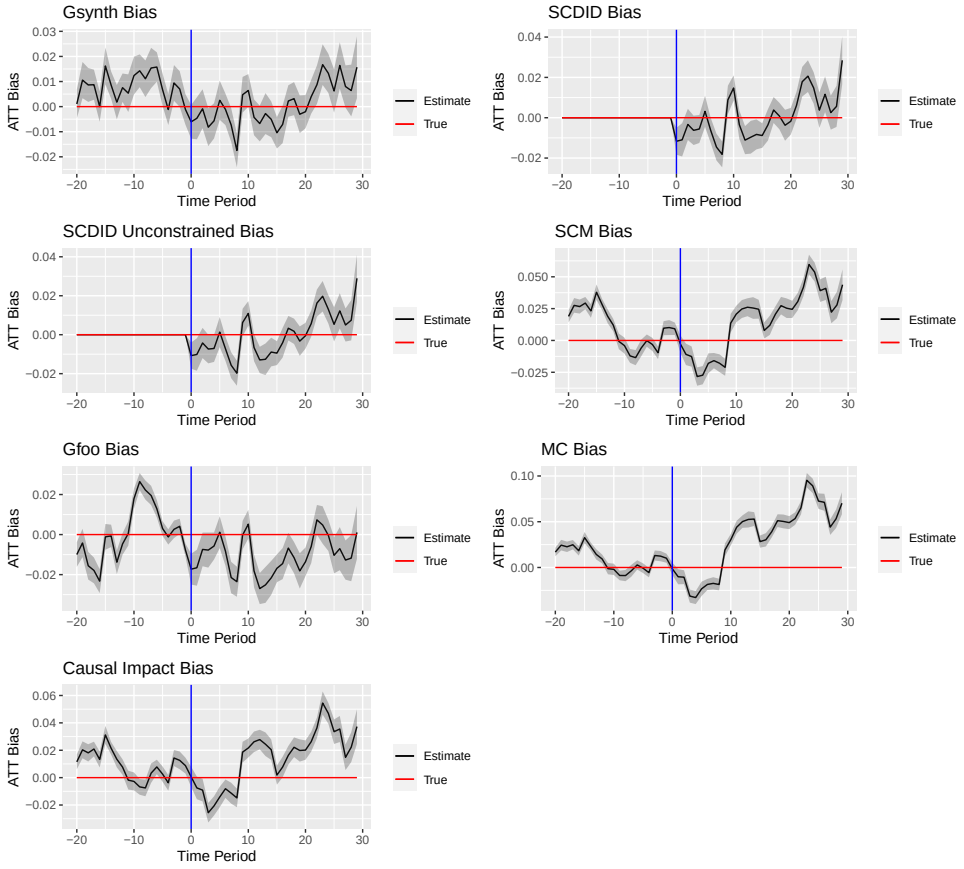
aa_noisy_factors_load_shift

Method	gsynth	scdid	scdid_uncon	scm	gfoo	mc	causalimpact
coverage							

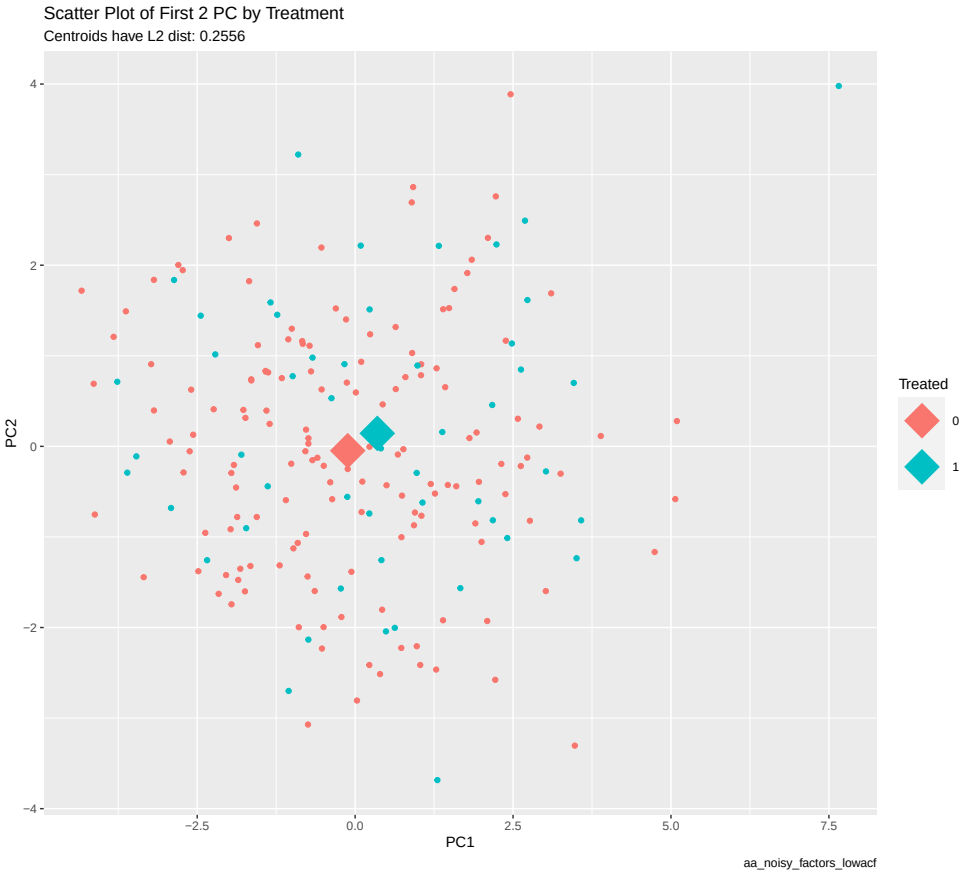
0	0.987	0.520	0.667	0.467	0.627	0.000	0.773
1	0.987	0.480	0.693	0.600	0.747	0.000	0.853
2	0.933	0.520	0.760	0.653	0.707	0.000	0.907
3	0.907	0.347	0.640	0.533	0.693	0.000	0.893
4	0.947	0.253	0.693	0.627	0.787	0.000	0.987
rmse							
0	0.236	0.261	0.239	0.242	0.280	0.490	0.257
1	0.250	0.285	0.258	0.258	0.304	0.588	0.265
2	0.252	0.301	0.268	0.255	0.336	0.686	0.266
3	0.266	0.349	0.302	0.268	0.373	0.843	0.273
4	0.272	0.371	0.316	0.273	0.413	0.935	0.278
bias							
0	0.004	0.071	0.048	0.066	0.059	0.300	0.043
1	−0.004	0.082	0.057	0.067	0.062	0.383	0.038
2	−0.020	0.083	0.051	0.060	0.066	0.465	0.024
3	−0.022	0.109	0.068	0.065	0.079	0.583	0.025
4	−0.026	0.117	0.072	0.062	0.087	0.670	0.013

Notes:

```
## [1] "aa_noisy_factors_lowacf"
```




```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
## # A tibble: 9 x 8
##   vars          n1    n2 statistic    df      p p.adj p.adj.signif
##   <chr>      <int> <int>     <dbl> <dbl>  <dbl> <dbl> <chr>
## 1 curvature    150    50   -0.552   71.7 0.583  0.750 ns
## 2 diff1_acf1   150    50   -0.627   85.9 0.532  0.750 ns
## 3 diff2_acf1   150    50    0.0828  88.7 0.934  0.934 ns
## 4 e_acf1       150    50   -0.558   78.5 0.579  0.750 ns
## 5 entropy      150    50    1.98    59.6 0.0518 0.466 ns
## 6 linearity    150    50   -0.357   78.6 0.722  0.812 ns
## 7 spike        150    50    0.640   75.3 0.524  0.750 ns
## 8 trend        150    50   -1.47    70.8 0.145  0.612 ns
## 9 x_acf1       150    50   -1.28    73.9 0.204  0.612 ns
```

Metrics by Method

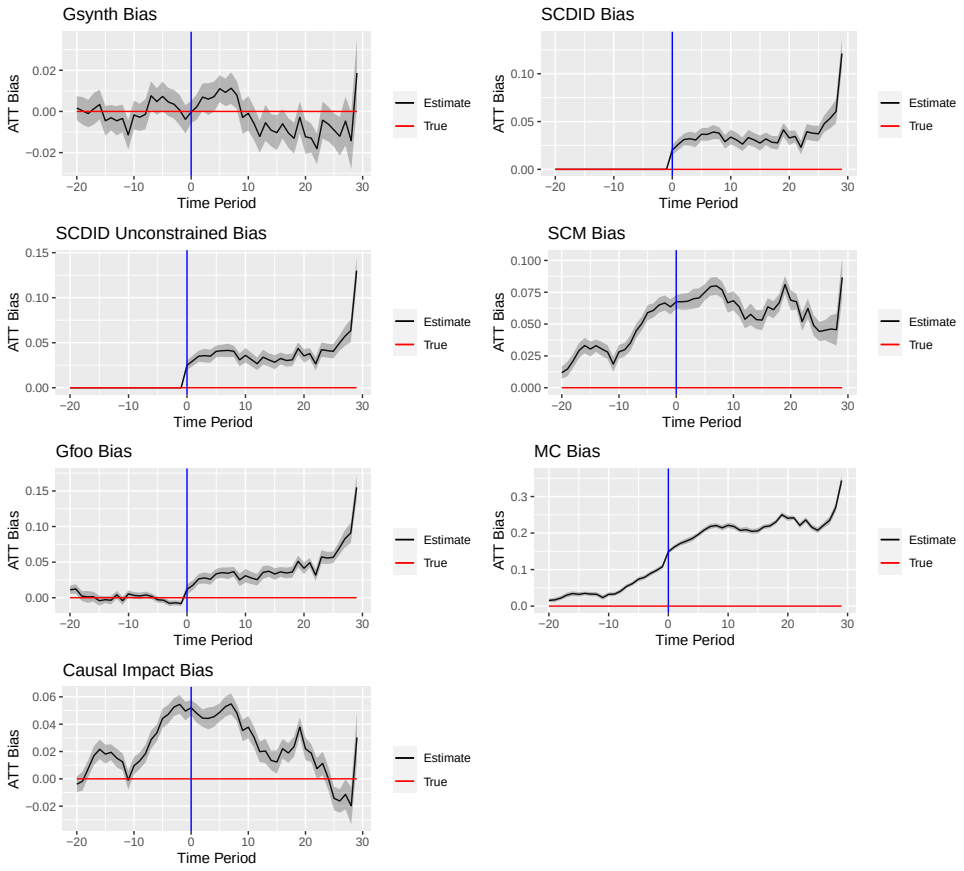
aa_noisy_factors_lowacf

Method	gsynth	scdid	scdid_uncon	scm	gfoo	mc	causalimpact
coverage							

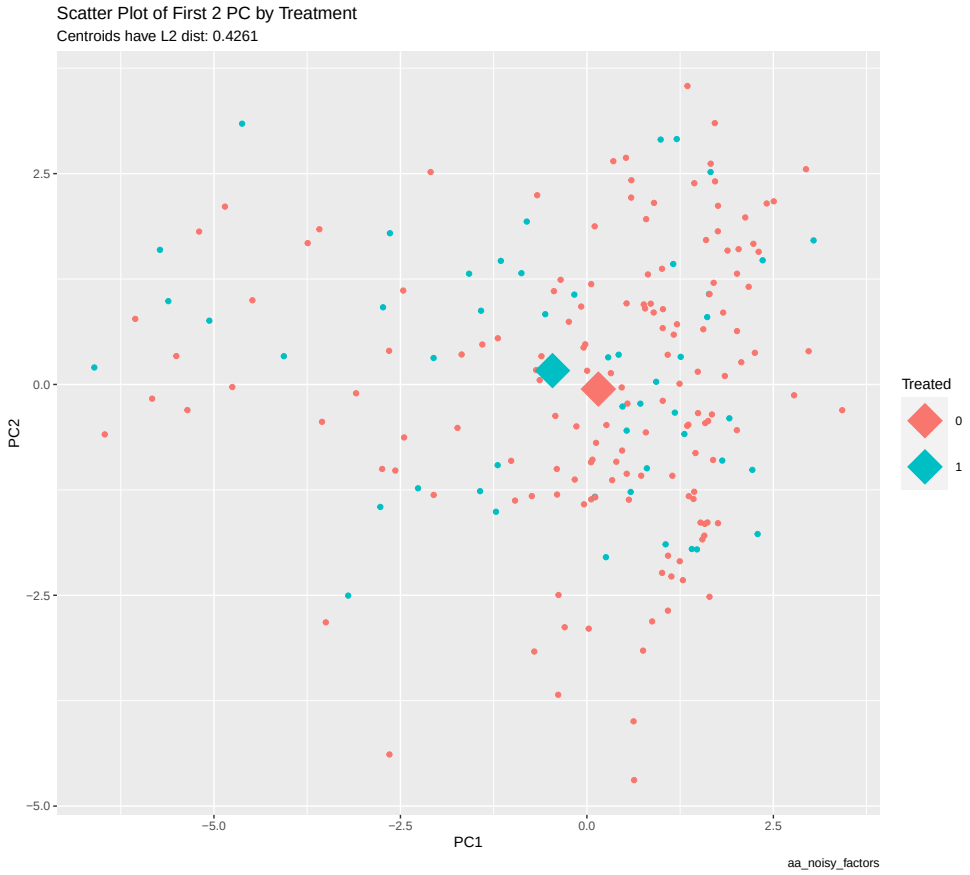
0	0.933	0.907	0.893	0.920	0.907	0.933	0.933
1	0.840	0.867	0.840	0.880	0.880	0.867	0.880
2	0.947	0.933	0.933	0.867	0.933	0.880	0.907
3	0.907	0.933	0.920	0.827	0.867	0.787	0.907
4	0.960	0.960	0.947	0.853	1.000	0.800	0.933
rmse							
0	0.209	0.209	0.208	0.213	0.243	0.211	0.223
1	0.212	0.213	0.211	0.218	0.245	0.215	0.228
2	0.210	0.212	0.210	0.215	0.242	0.214	0.224
3	0.207	0.211	0.210	0.216	0.236	0.215	0.224
4	0.212	0.215	0.213	0.221	0.237	0.222	0.231
bias							
0	−0.006	−0.012	−0.011	−0.003	−0.017	−0.001	0.001
1	−0.005	−0.011	−0.010	−0.011	−0.017	−0.010	−0.008
2	−0.001	−0.003	−0.004	−0.013	−0.007	−0.011	−0.009
3	−0.008	−0.006	−0.007	−0.028	−0.008	−0.031	−0.026
4	−0.006	−0.006	−0.007	−0.027	−0.006	−0.033	−0.021

Notes:

```
## [1] "aa_noisy_factors"
```




```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
## # A tibble: 9 x 8
##   vars          n1    n2 statistic    df      p p.adj p.adj.signif
##   <chr>      <int> <int>     <dbl> <dbl>  <dbl> <dbl> <chr>
## 1 curvature    150    50   0.00564  86.3 0.996  0.996 ns
## 2 diff1_acf1   150    50   0.992    98.3 0.324  0.486 ns
## 3 diff2_acf1   150    50   0.548    90.1 0.585  0.752 ns
## 4 e_acf1       150    50   0.368    87.4 0.714  0.803 ns
## 5 entropy      150    50   1.40     69.2 0.167  0.301 ns
## 6 linearity    150    50  -1.90     75.5 0.0612 0.295 ns
## 7 spike        150    50   1.64     76.8 0.105  0.295 ns
## 8 trend        150    50  -1.57     73.1 0.12   0.295 ns
## 9 x_acf1       150    50  -1.53     75.2 0.131  0.295 ns
```

Metrics by Method

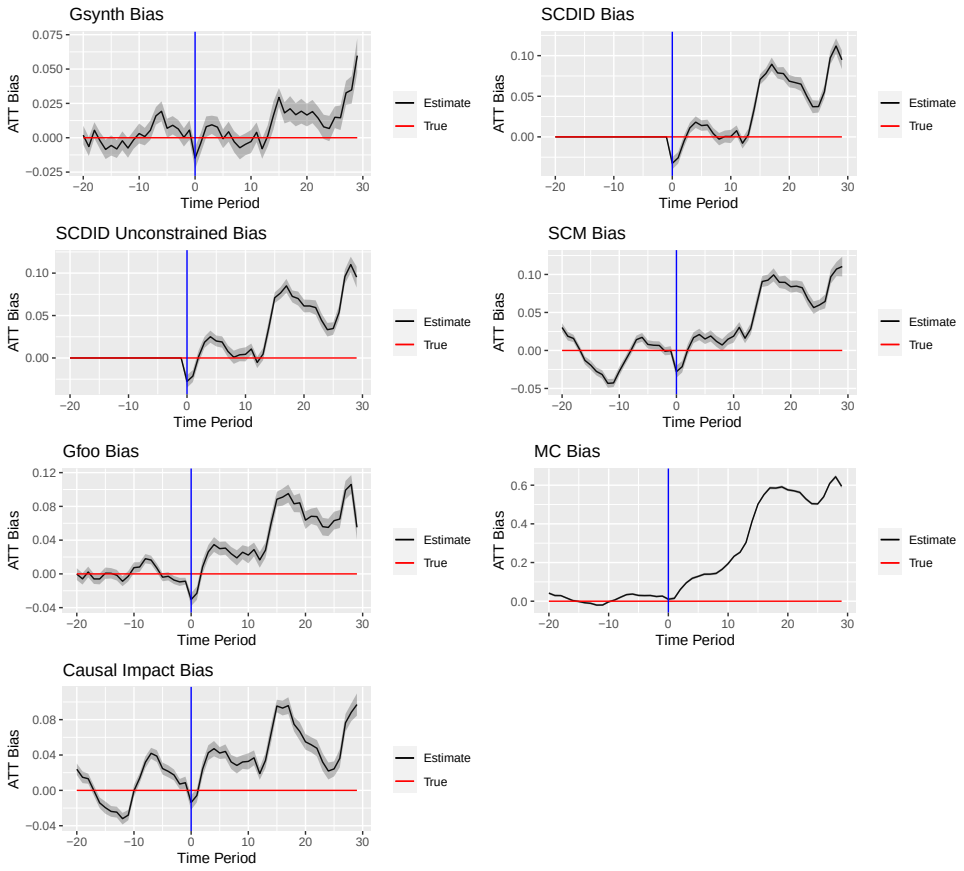
aa_noisy_factors

Method	gsynth	scdid	scdid_uncon	scm	gfoo	mc	causalimpact
coverage							

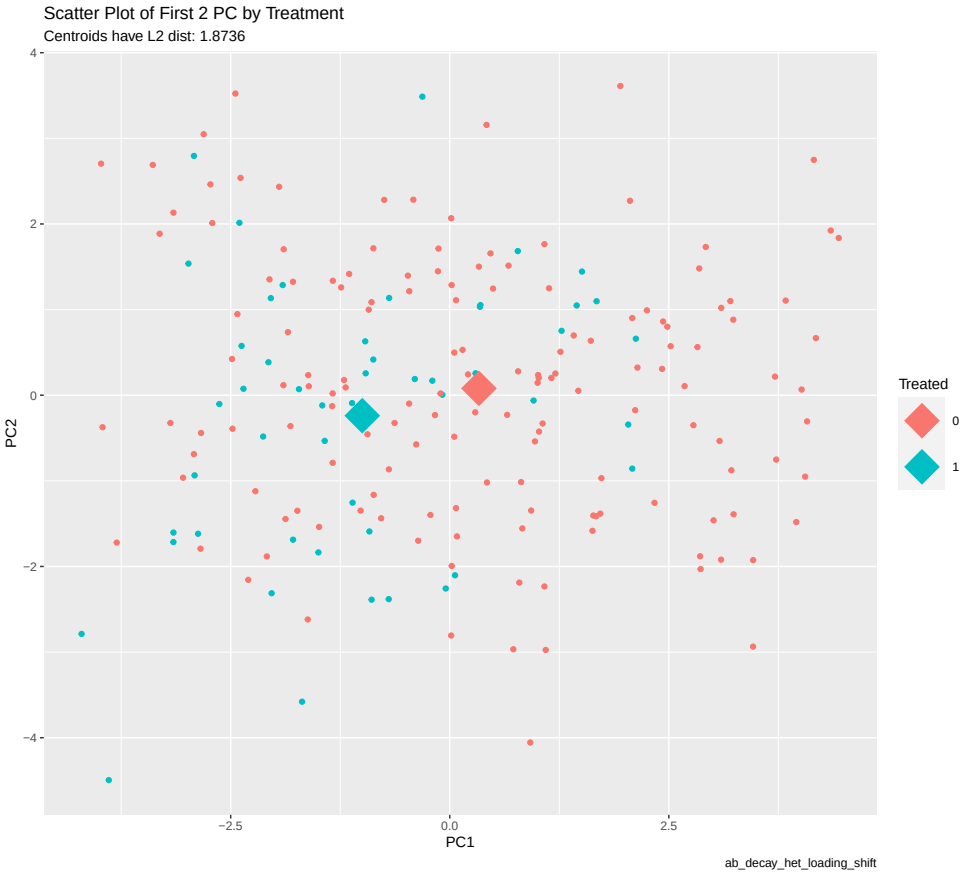
0	0.987	0.960	0.947	0.360	0.947	0.000	0.667
1	0.947	0.960	0.920	0.440	0.933	0.013	0.693
2	0.987	0.840	0.800	0.467	0.920	0.000	0.707
3	0.920	0.893	0.787	0.400	0.853	0.000	0.693
4	0.947	0.813	0.773	0.427	0.907	0.000	0.693
rmse							
0	0.213	0.227	0.223	0.228	0.232	0.301	0.226
1	0.221	0.234	0.231	0.238	0.242	0.327	0.235
2	0.216	0.231	0.228	0.232	0.241	0.331	0.228
3	0.214	0.233	0.228	0.230	0.243	0.340	0.226
4	0.217	0.233	0.229	0.233	0.248	0.344	0.229
bias							
0	−0.000	0.020	0.025	0.067	0.012	0.149	0.052
1	0.002	0.026	0.030	0.068	0.018	0.162	0.048
2	0.007	0.031	0.035	0.068	0.026	0.172	0.044
3	0.006	0.032	0.036	0.070	0.028	0.178	0.044
4	0.007	0.031	0.035	0.071	0.026	0.185	0.045

Notes:

```
## [1] "ab_decay_het_loading_shift"
```




```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
## # A tibble: 9 x 8
##   vars          n1    n2 statistic    df          p      p.adj p.adj.signif
##   <chr>      <int> <int>     <dbl> <dbl>      <dbl>      <dbl> <chr>
## 1 curvature    150    50    -3.09  115.  0.00253    0.00569    **
## 2 diff1_acf1   150    50    -2.88   70.3 0.00522    0.00783    **
## 3 diff2_acf1   150    50     0.377  81.7 0.707      0.707      ns
## 4 e_acf1       150    50    -2.92   80.5 0.0045     0.00783    **
## 5 entropy      150    50     2.32  103.  0.0226     0.0291     *
## 6 linearity    150    50    -1.13  111.  0.26       0.292      ns
## 7 spike        150    50     5.30  151.  0.0000004  0.0000036  ****
## 8 trend        150    50    -4.72  111.  0.00000702 0.0000211  ****
## 9 x_acf1       150    50    -4.99  113.  0.00000218 0.00000981 ****
```

Metrics by Method

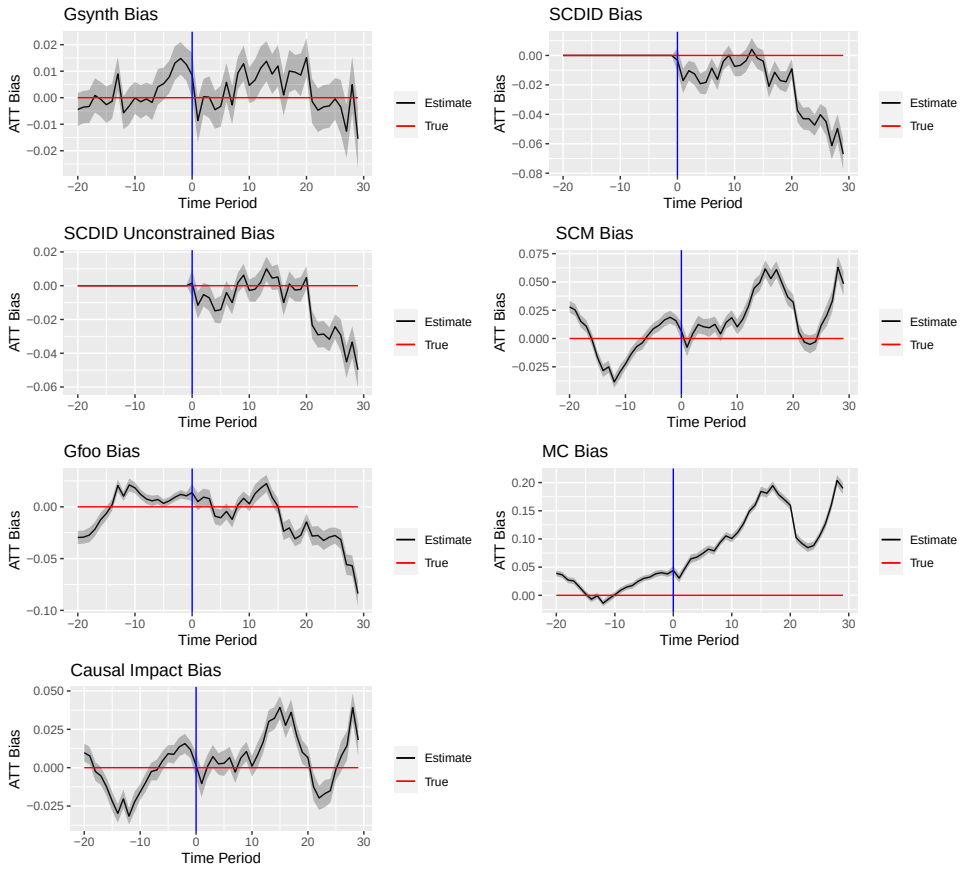
ab_decay_het_loading_shift

Method	gsynth	scdid	scdid_uncon	scm	gfoo	mc	causalimpact
coverage							

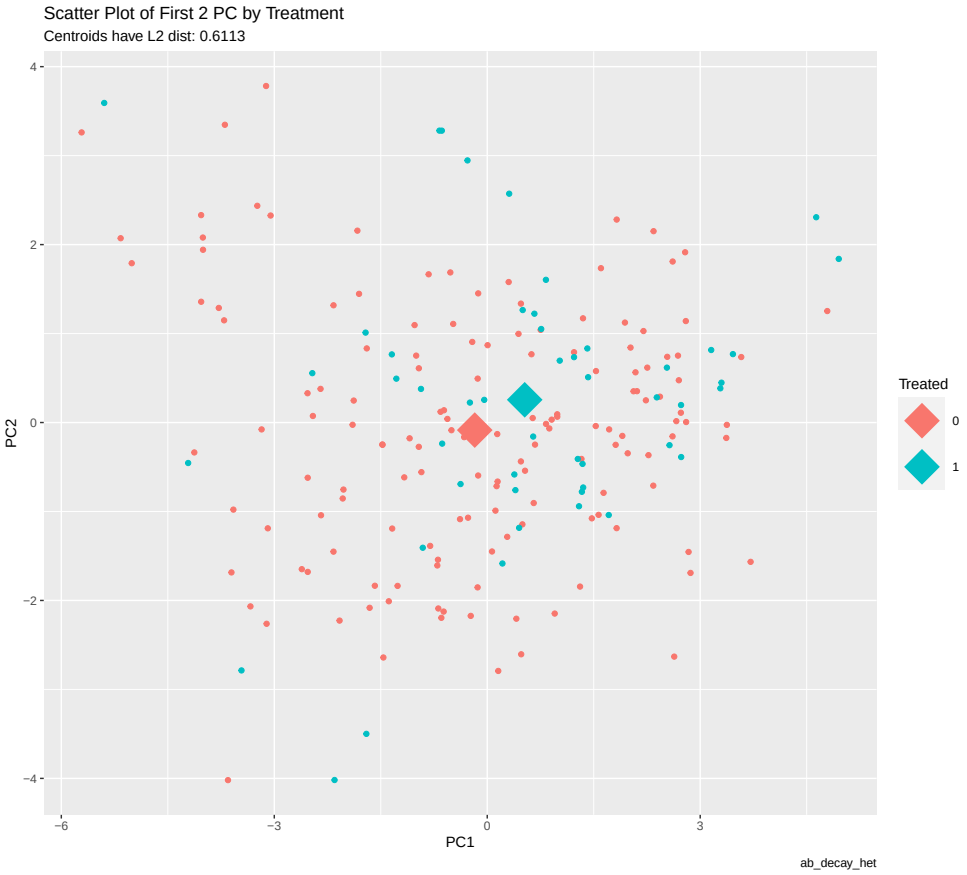
0	0.933	0.867	0.880	0.867	0.867	0.960	0.933
1	0.960	0.947	0.973	0.920	0.973	0.920	0.987
2	0.947	0.987	0.987	0.987	0.960	0.680	0.880
3	0.973	0.973	0.947	0.947	0.933	0.187	0.760
4	0.933	0.933	0.880	0.920	0.893	0.133	0.760
rmse							
0	0.220	0.239	0.231	0.229	0.250	0.241	0.241
1	0.223	0.243	0.236	0.229	0.260	0.255	0.245
2	0.219	0.250	0.241	0.232	0.271	0.270	0.246
3	0.219	0.249	0.238	0.229	0.280	0.280	0.245
4	0.220	0.255	0.242	0.233	0.292	0.300	0.252
bias							
0	−0.015	−0.033	−0.028	−0.028	−0.030	0.010	−0.014
1	−0.004	−0.026	−0.021	−0.021	−0.023	0.015	−0.006
2	0.008	−0.005	0.002	0.001	0.008	0.061	0.024
3	0.009	0.011	0.019	0.017	0.026	0.095	0.043
4	0.008	0.018	0.025	0.021	0.035	0.118	0.047

Notes:

```
## [1] "ab_decay_het"
```




```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
## # A tibble: 9 x 8
##   vars          n1    n2 statistic    df       p  p.adj p.adj.signif
##   <chr>      <int> <int>     <dbl> <dbl>   <dbl>  <dbl> <chr>
## 1 curvature    150    50    -0.117  87.7 0.907   0.907  ns
## 2 diff1_acf1   150    50     2.28   78.9 0.025   0.112  ns
## 3 diff2_acf1   150    50     1.69   79.7 0.0945  0.170  ns
## 4 e_acf1       150    50     2.73   79.3 0.00778 0.0700 ns
## 5 entropy      150    50    -1.45  102.  0.151   0.226  ns
## 6 linearity    150    50     1.93   85.8 0.057   0.170  ns
## 7 spike        150    50    -0.823  76.7 0.413   0.465  ns
## 8 trend        150    50     0.958  90.7 0.341   0.438  ns
## 9 x_acf1       150    50     1.74   85.9 0.0857  0.170  ns
```

Metrics by Method

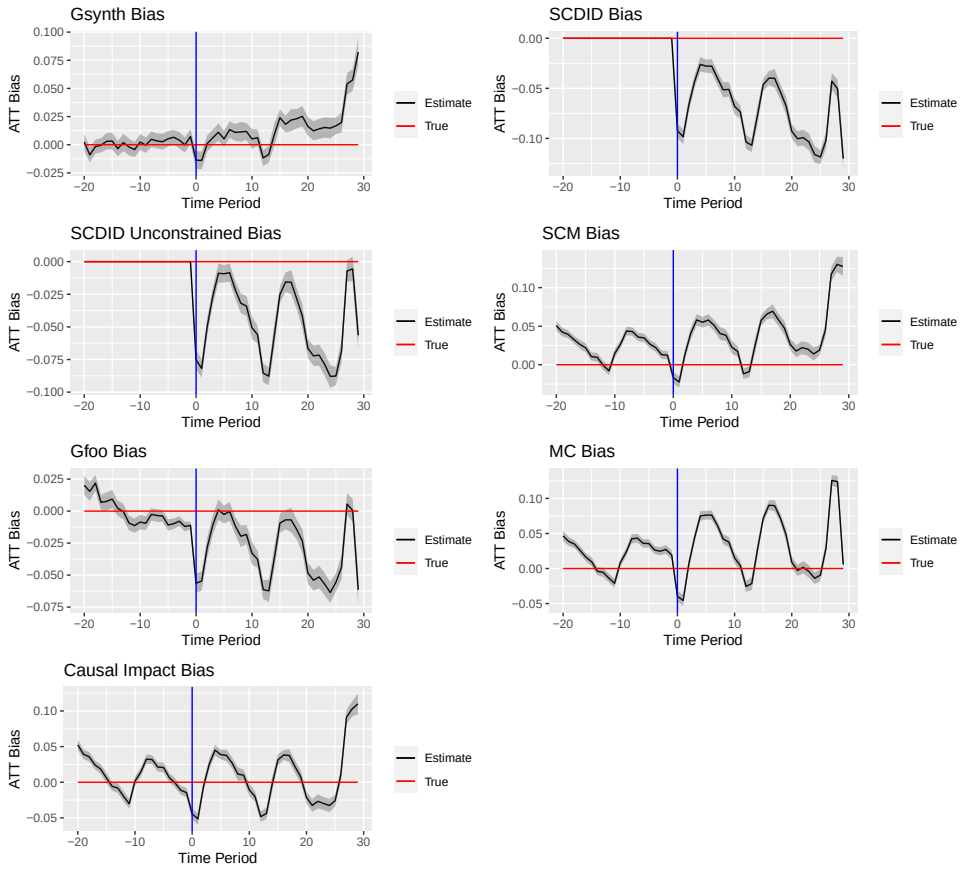
ab_decay_het

Method	gsynth	scdid	scdid_uncon	scm	gfoo	mc	causalimpact
coverage							

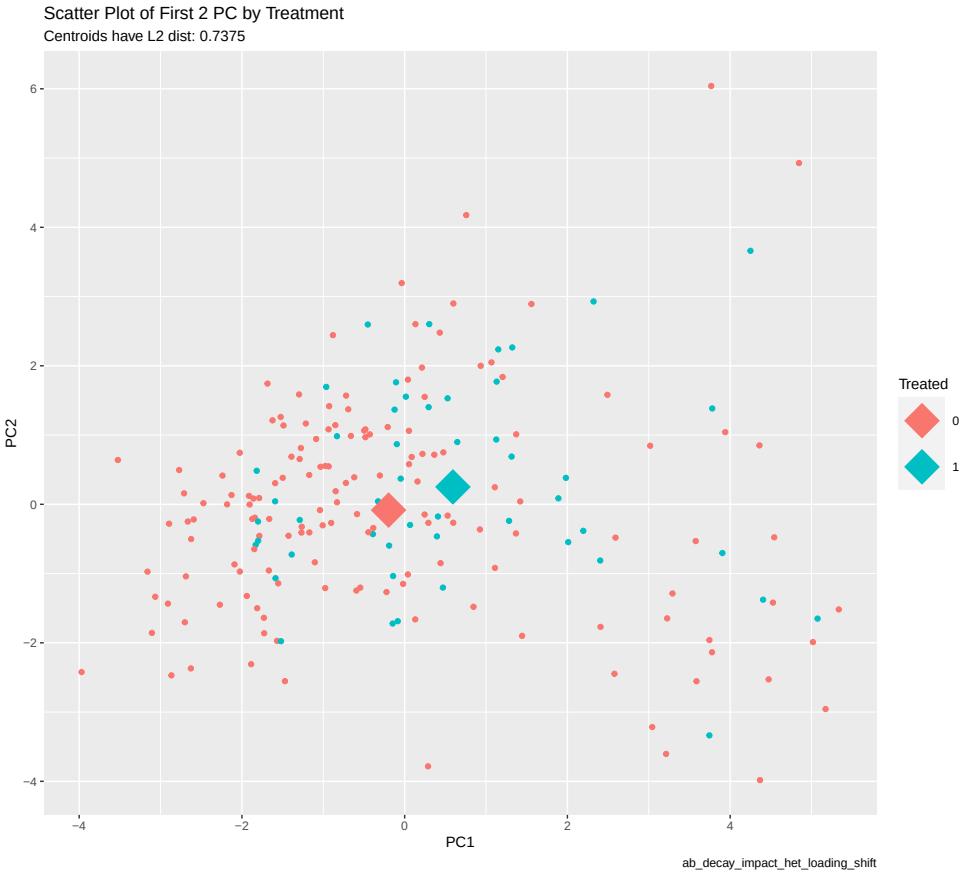
0	0.920	0.920	0.920	0.907	0.933	0.813	0.933
1	0.907	0.920	0.933	0.960	0.947	0.907	0.920
2	0.947	0.960	0.973	0.973	0.933	0.853	0.973
3	0.960	0.907	0.907	0.907	0.920	0.653	0.920
4	0.973	0.960	0.987	0.973	0.973	0.640	0.973
rmse							
0	0.217	0.240	0.236	0.230	0.240	0.272	0.236
1	0.223	0.245	0.242	0.232	0.242	0.276	0.237
2	0.219	0.244	0.242	0.232	0.240	0.278	0.236
3	0.222	0.244	0.243	0.233	0.236	0.286	0.235
4	0.223	0.249	0.249	0.234	0.239	0.298	0.238
bias							
0	0.008	−0.003	0.002	0.007	0.014	0.044	0.002
1	−0.009	−0.017	−0.012	−0.008	0.005	0.031	−0.010
2	0.000	−0.010	−0.005	0.004	0.009	0.048	0.001
3	0.000	−0.013	−0.007	0.012	0.008	0.065	0.007
4	−0.005	−0.019	−0.015	0.010	−0.009	0.068	0.003

Notes:

```
## [1] "ab_decay_impact_het_loading_shift"
```




```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
## # A tibble: 9 x 8
##   vars          n1    n2 statistic    df        p   p.adj p.adj.signif
##   <chr>      <int> <int>     <dbl> <dbl>    <dbl>   <dbl> <chr>
## 1 curvature    150    50     0.373 137.   0.71     0.71    ns
## 2 diff1_acf1   150    50    -2.20   90.2 0.0302   0.0906  ns
## 3 diff2_acf1   150    50    -0.590  87.4 0.557    0.627   ns
## 4 e_acf1       150    50    -3.74   95.8 0.000309 0.00278 **
## 5 entropy      150    50     1.20   81.8 0.234    0.301   ns
## 6 linearity    150    50    -1.47   80.6 0.146    0.219   ns
## 7 spike        150    50     2.02  102.  0.0464   0.104   ns
## 8 trend        150    50    -1.85   91.1 0.0669   0.120   ns
## 9 x_acf1       150    50    -2.59   94.4 0.0112   0.0504  ns
```

Metrics by Method

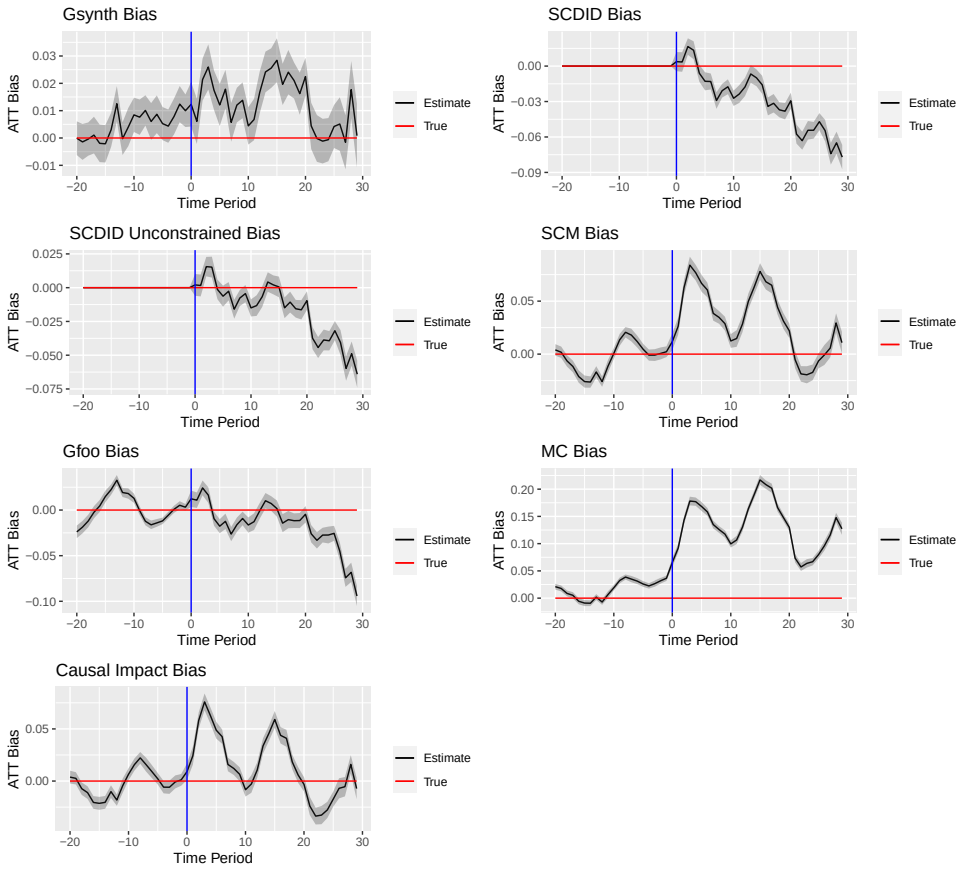
ab_decay_impact_het_loading_shift

Method	gsynth	scdid	scdid_uncon	scm	gfoo	mc	causalimpact
coverage							

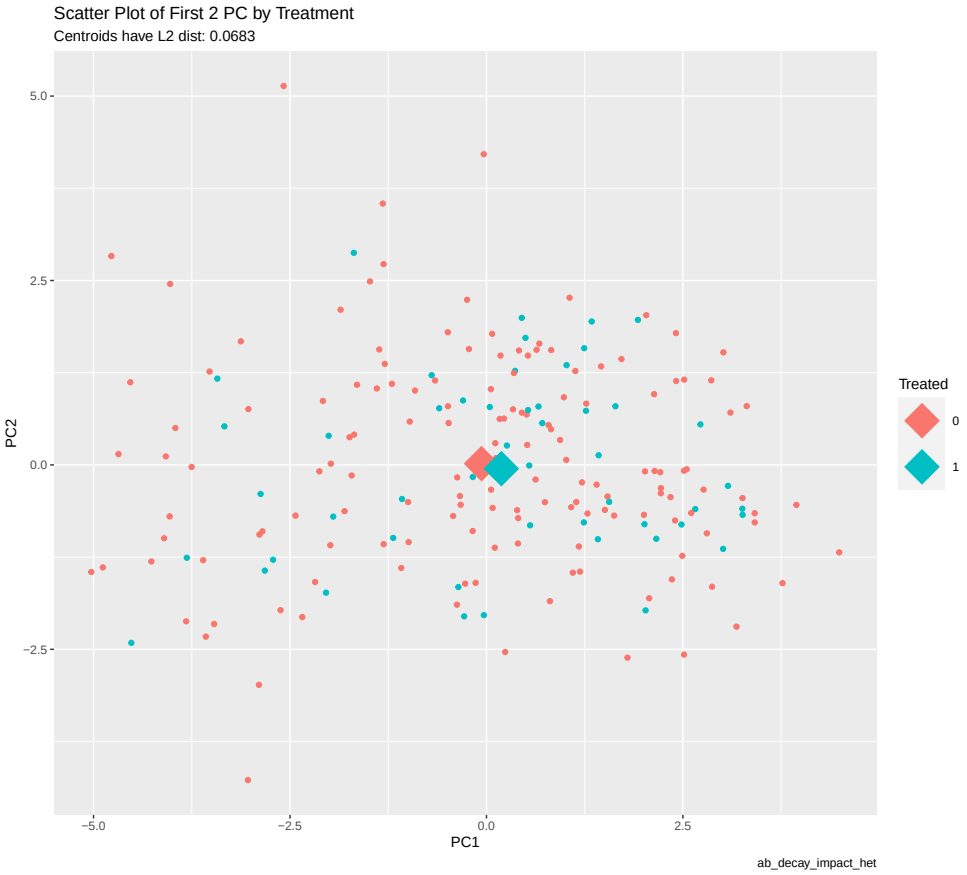
0	0.947	0.800	0.680	0.947	0.653	1.000	0.787
1	0.947	0.907	0.773	0.907	0.667	1.000	0.733
2	0.973	0.987	0.920	0.947	0.920	1.000	0.960
3	0.973	1.000	0.973	0.773	0.960	1.000	0.893
4	0.920	1.000	0.987	0.587	0.987	1.000	0.813
rmse							
0	0.230	0.424	0.326	0.227	0.256	0.536	0.247
1	0.226	0.493	0.383	0.236	0.259	0.618	0.257
2	0.236	0.465	0.362	0.237	0.275	0.642	0.254
3	0.231	0.434	0.335	0.228	0.275	0.634	0.243
4	0.236	0.410	0.311	0.235	0.284	0.629	0.249
bias							
0	−0.014	−0.092	−0.075	−0.017	−0.056	−0.040	−0.044
1	−0.014	−0.098	−0.082	−0.022	−0.055	−0.046	−0.051
2	0.001	−0.066	−0.050	0.014	−0.028	0.007	−0.005
3	0.006	−0.043	−0.026	0.041	−0.011	0.044	0.025
4	0.011	−0.026	−0.009	0.058	0.001	0.075	0.045

Notes:

```
## [1] "ab_decay_impact_het"
```




```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
## # A tibble: 9 x 8
##   vars          n1    n2 statistic    df     p p.adj p.adj.signif
##   <chr>      <int> <int>     <dbl> <dbl> <dbl> <dbl> <chr>
## 1 curvature    150    50     0.332  88.8 0.74  0.912 ns
## 2 diff1_acf1   150    50     0.748  99.0 0.456 0.912 ns
## 3 diff2_acf1   150    50     0.110  99.7 0.912 0.912 ns
## 4 e_acf1       150    50     1.18   79.6 0.24  0.912 ns
## 5 entropy      150    50    -0.992  84.5 0.324 0.912 ns
## 6 linearity    150    50     1.20   78.4 0.233 0.912 ns
## 7 spike        150    50    -0.229  85.8 0.82  0.912 ns
## 8 trend        150    50     0.221  89.4 0.825 0.912 ns
## 9 x_acf1       150    50     0.471  88.7 0.639 0.912 ns
```

Metrics by Method

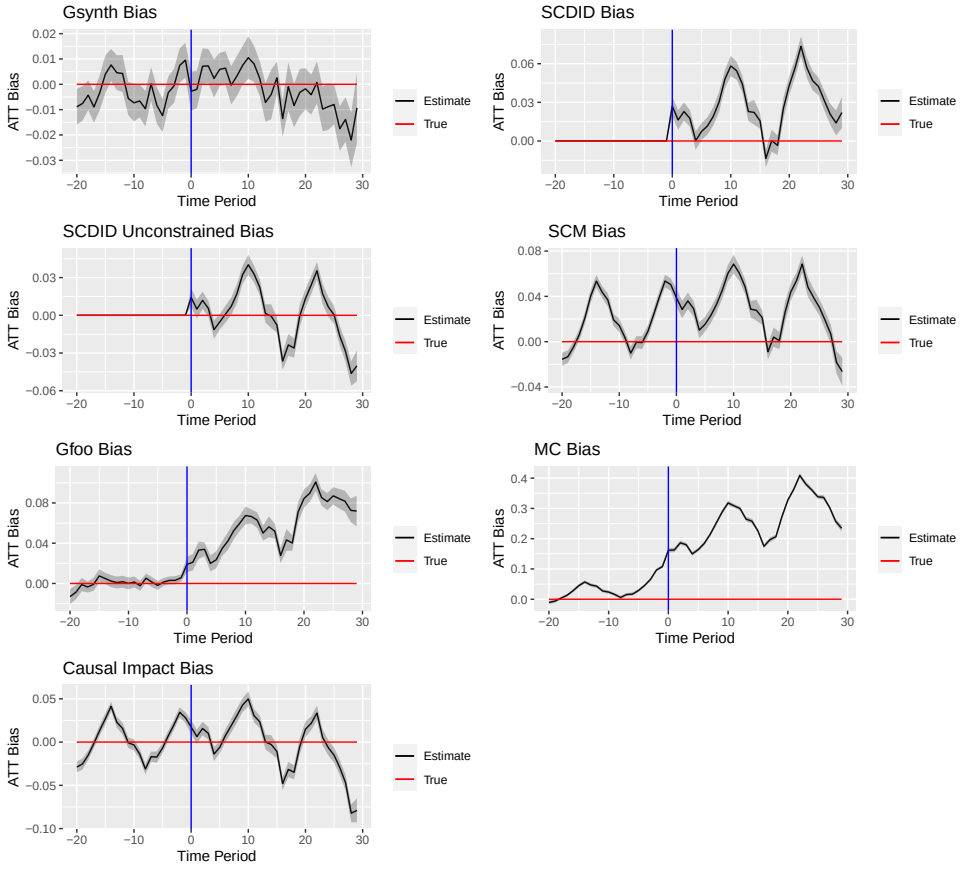
ab_decay_impact_het

Method	gsynth	scdid	scdid_uncon	scm	gfoo	mc	causalimpact
coverage							

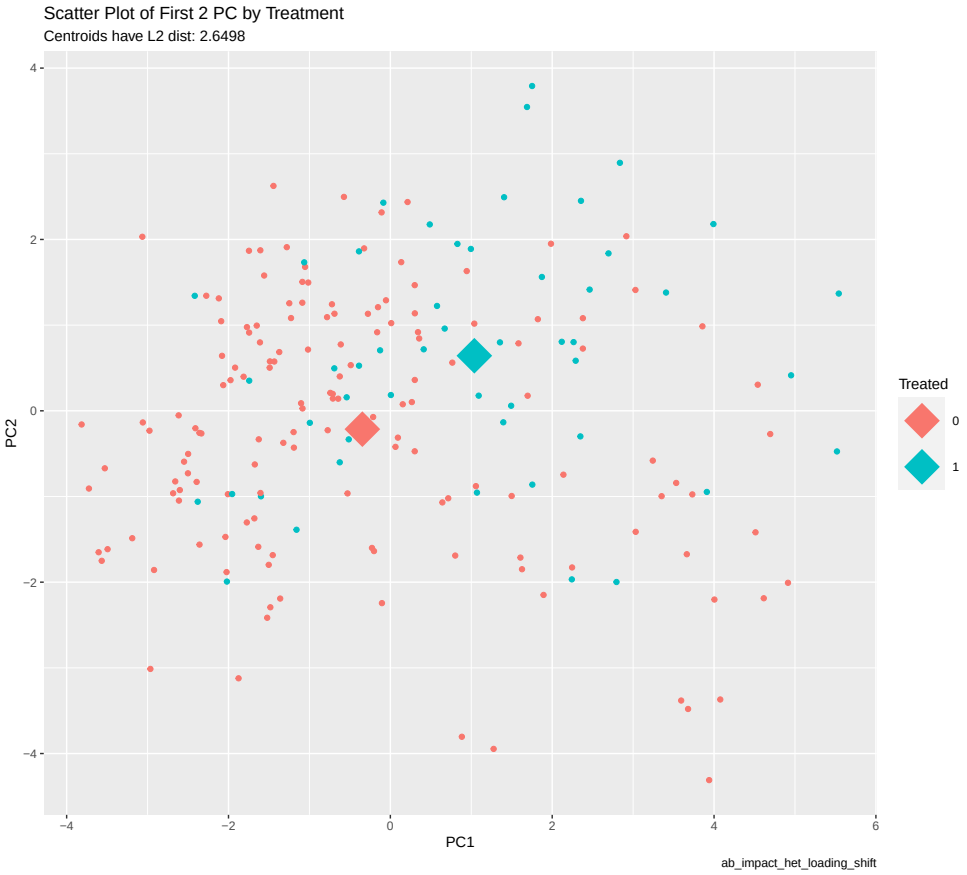
0	0.920	0.947	0.947	0.880	0.907	0.667	0.933
1	0.920	0.960	0.973	0.853	0.947	0.387	0.853
2	0.867	1.000	0.987	0.507	0.933	0.027	0.587
3	0.867	0.973	0.947	0.333	0.933	0.000	0.413
4	0.920	0.987	0.987	0.387	0.987	0.000	0.547
rmse							
0	0.221	0.243	0.238	0.233	0.243	0.276	0.239
1	0.229	0.254	0.249	0.240	0.260	0.287	0.245
2	0.229	0.256	0.252	0.249	0.269	0.309	0.254
3	0.234	0.258	0.254	0.259	0.277	0.332	0.261
4	0.232	0.259	0.254	0.251	0.277	0.330	0.250
bias							
0	0.012	0.004	0.002	0.012	0.012	0.065	0.009
1	0.006	0.003	0.002	0.026	0.011	0.092	0.024
2	0.021	0.017	0.016	0.063	0.024	0.143	0.058
3	0.026	0.013	0.015	0.084	0.016	0.178	0.076
4	0.017	−0.006	−0.001	0.077	−0.009	0.177	0.063

Notes:

```
## [1] "ab_impact_het_loading_shift"
```




```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
## # A tibble: 9 x 8
##   vars          n1    n2 statistic    df           p       p.adj p.adj.signif
##   <chr>      <int> <int>     <dbl> <dbl>       <dbl>       <dbl> <chr>
## 1 curvature    150    50      1.45 129.  0.149        0.149       ns
## 2 diff1_acf1   150    50     -5.49  71.7 0.000000566  0.00000255  ****
## 3 diff2_acf1   150    50     -2.53  77.1 0.0134       0.0201      *
## 4 e_acf1       150    50     -6.17  79.3 0.0000000274 0.000000247 ****
## 5 entropy      150    50      2.41 100.  0.0178       0.0229      *
## 6 linearity    150    50     -1.68  98.6 0.0958       0.108       ns
## 7 spike        150    50      3.70 119.  0.000334     0.000752    ***
## 8 trend        150    50     -2.63  95.6 0.00999      0.0180      *
## 9 x_acf1       150    50     -3.91  97.7 0.000172     0.000516    ***
```

Metrics by Method

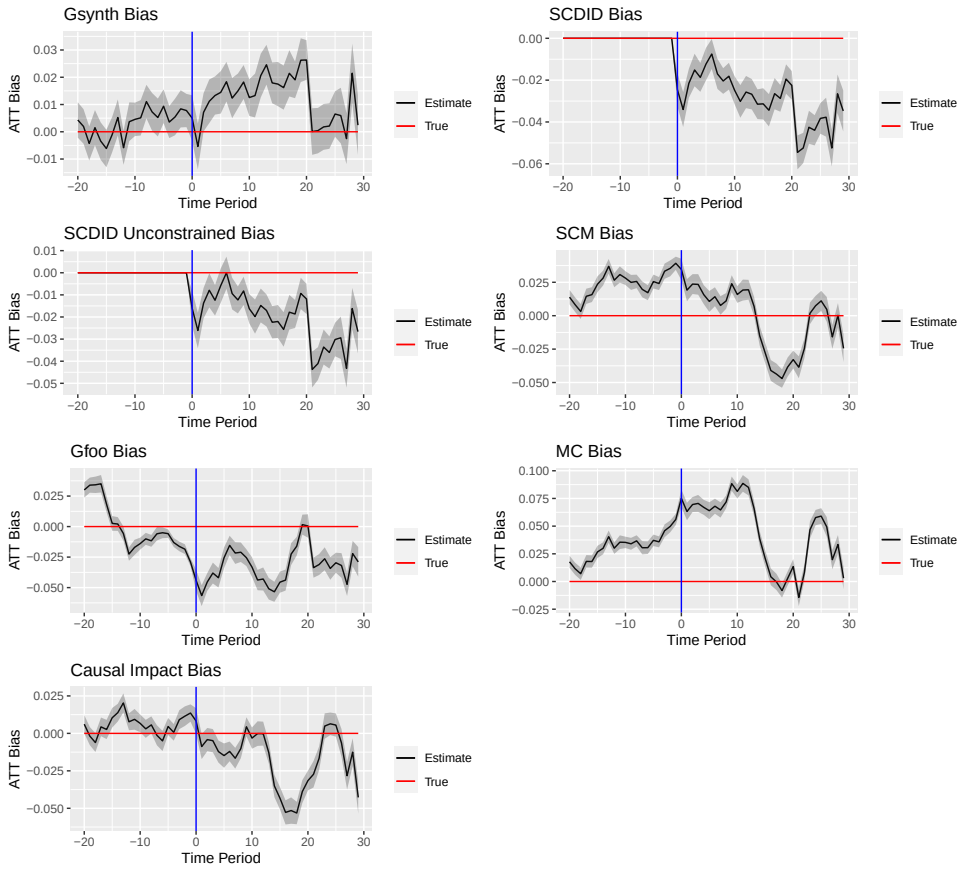
ab_impact_het_loading_shift

Method	gsynth	scdid	scdid_uncon	scm	gfoo	mc	causalimpact
coverage							

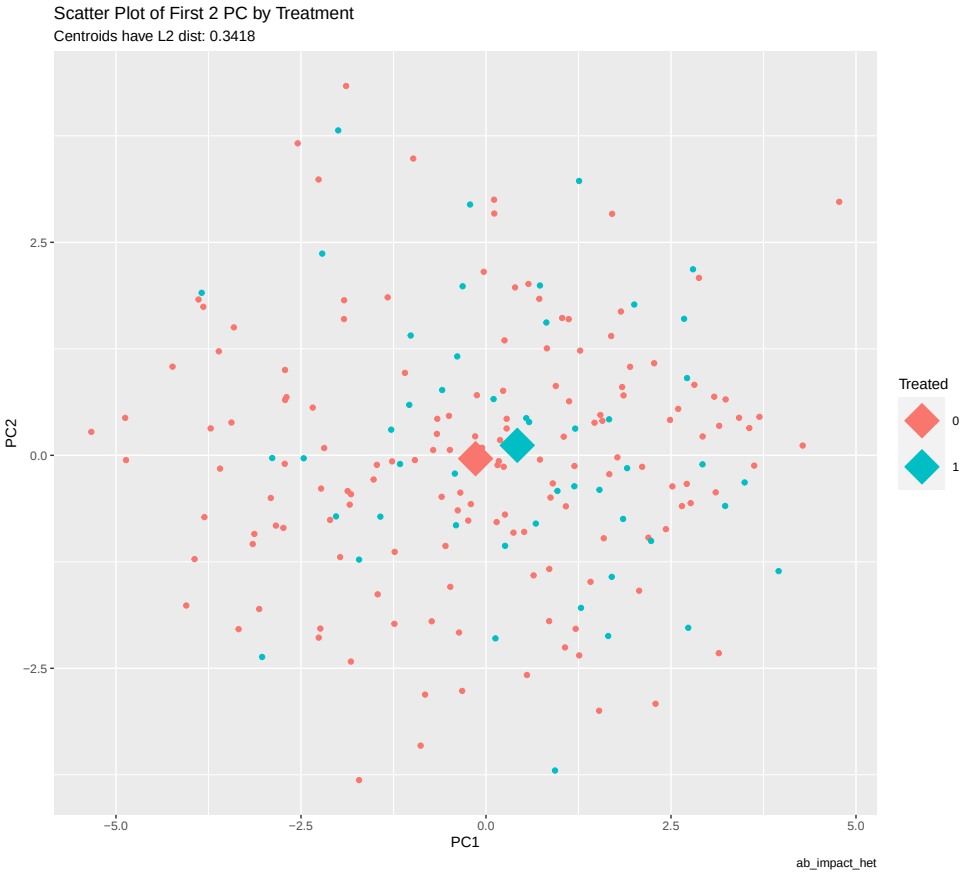
0	0.933	0.907	0.893	0.800	0.933	0.027	0.893
1	0.947	0.960	0.973	0.853	0.920	0.013	0.933
2	0.920	0.933	0.933	0.800	0.800	0.000	0.893
3	0.987	0.987	0.987	0.880	0.933	0.027	0.960
4	0.880	0.987	0.933	0.907	0.920	0.093	0.933
rmse							
0	0.212	0.250	0.220	0.225	0.243	0.375	0.236
1	0.217	0.258	0.224	0.226	0.255	0.395	0.233
2	0.215	0.262	0.223	0.225	0.256	0.438	0.236
3	0.215	0.264	0.224	0.227	0.261	0.446	0.237
4	0.218	0.259	0.224	0.225	0.266	0.419	0.237
bias							
0	−0.003	0.027	0.014	0.039	0.019	0.162	0.017
1	−0.002	0.016	0.005	0.028	0.021	0.163	0.006
2	0.007	0.023	0.012	0.036	0.033	0.186	0.016
3	0.007	0.018	0.006	0.029	0.034	0.180	0.010
4	0.002	0.000	−0.011	0.010	0.020	0.150	−0.014

Notes:

```
## [1] "ab_impact_het"
```




```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
## # A tibble: 9 x 8
##   vars          n1    n2 statistic    df       p  p.adj p.adj.signif
##   <chr>      <int> <int>     <dbl> <dbl>   <dbl>  <dbl> <chr>
## 1 curvature    150    50     0.580  83.7 0.563   0.625  ns
## 2 diff1_acf1   150    50     1.39   94.7 0.169   0.380  ns
## 3 diff2_acf1   150    50     0.491 102.  0.625   0.625  ns
## 4 e_acf1       150    50     1.95   78.9 0.0545  0.235  ns
## 5 entropy      150    50    -1.78  119.  0.0782  0.235  ns
## 6 linearity    150    50     2.88   82.2 0.00511 0.0460 *
## 7 spike        150    50    -1.01   79.8 0.315   0.514  ns
## 8 trend        150    50     0.846  89.3 0.4     0.514  ns
## 9 x_acf1       150    50     0.923  96.1 0.358   0.514  ns
```

Metrics by Method

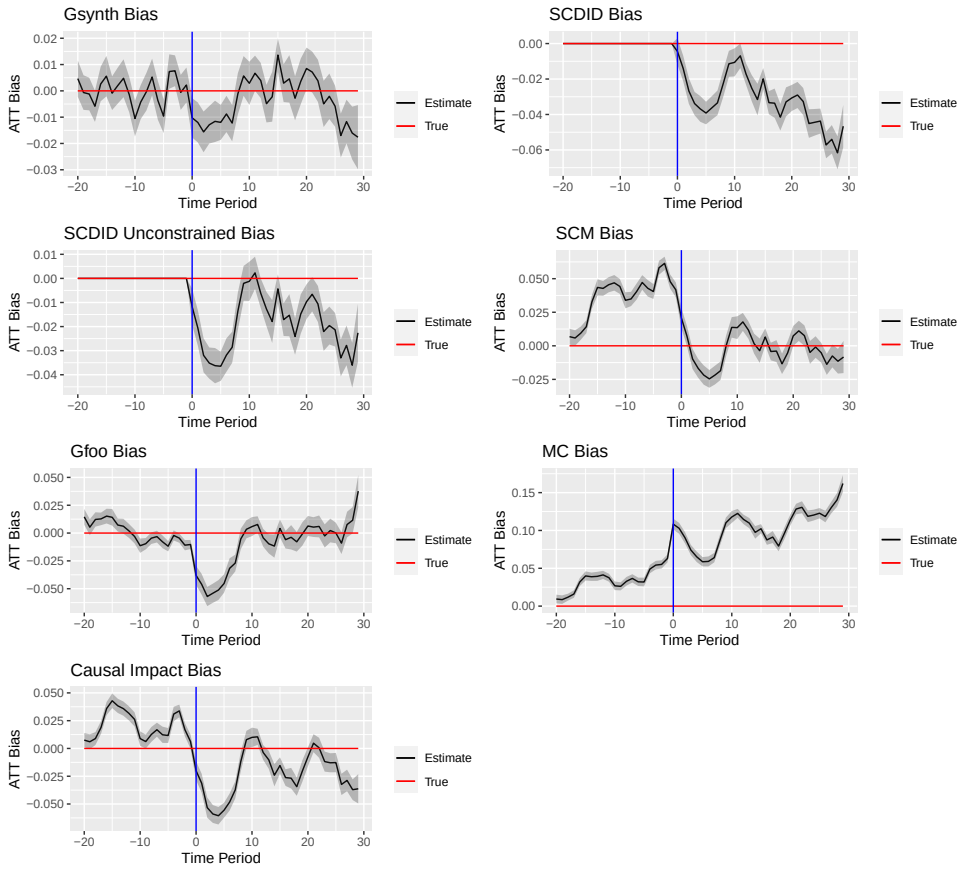
ab_impact_het

Method	gsynth	scdid	scdid_uncon	scm	gfoo	mc	causalimpact
coverage							

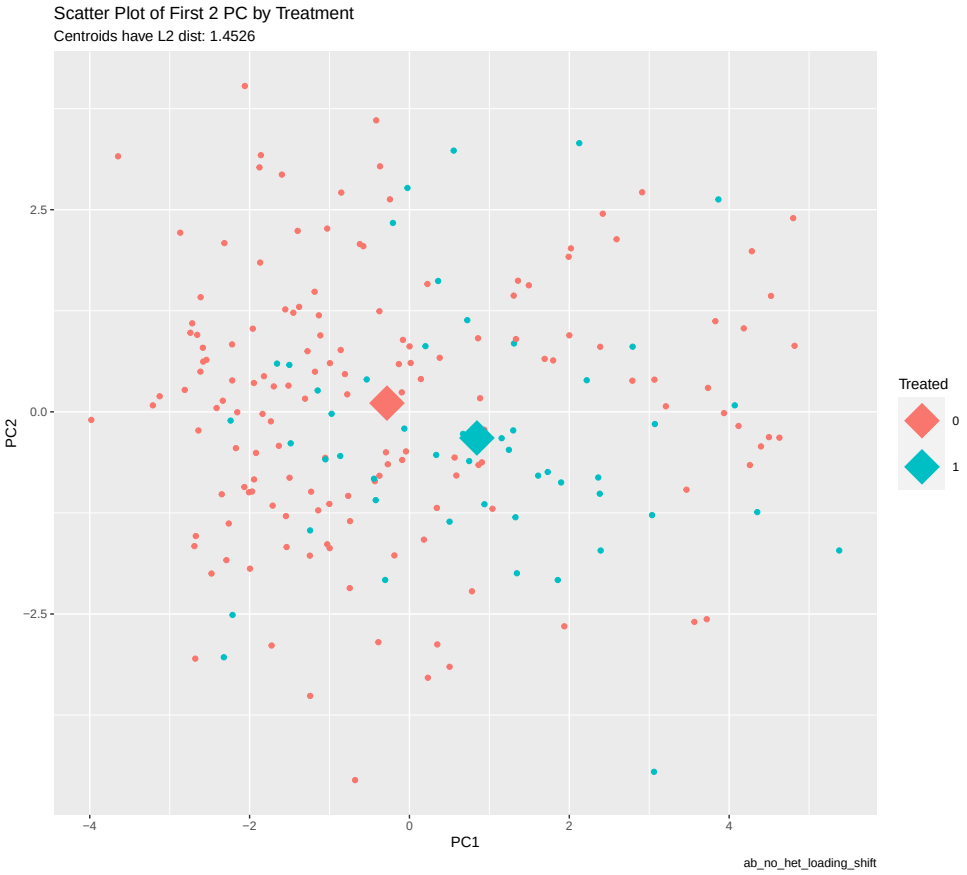
0	0.920	0.867	0.893	0.787	0.800	0.560	0.907
1	0.920	0.787	0.867	0.867	0.587	0.667	0.907
2	0.960	0.893	0.933	0.893	0.773	0.667	0.907
3	0.907	0.907	0.933	0.867	0.813	0.573	0.947
4	0.947	0.947	0.960	0.960	0.787	0.747	0.960
rmse							
0	0.217	0.231	0.227	0.226	0.247	0.284	0.229
1	0.224	0.232	0.229	0.227	0.256	0.287	0.232
2	0.218	0.227	0.225	0.224	0.251	0.292	0.227
3	0.220	0.226	0.222	0.225	0.251	0.294	0.229
4	0.225	0.233	0.227	0.230	0.264	0.311	0.233
bias							
0	0.005	−0.025	−0.016	0.035	−0.044	0.075	0.008
1	−0.005	−0.034	−0.026	0.019	−0.057	0.063	−0.009
2	0.007	−0.022	−0.014	0.024	−0.045	0.070	−0.004
3	0.011	−0.015	−0.008	0.023	−0.038	0.071	−0.005
4	0.013	−0.019	−0.012	0.016	−0.042	0.067	−0.012

Notes:

```
## [1] "ab_no_het_loading_shift"
```




```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
## # A tibble: 9 x 8
##   vars          n1    n2 statistic    df         p    p.adj p.adj.signif
##   <chr>      <int> <int>     <dbl> <dbl>     <dbl>    <dbl> <chr>
## 1 curvature    150    50     3.77  102.  0.000278  0.000626 ***
## 2 diff1_acf1   150    50    -2.16   85.7 0.0338    0.0608   ns
## 3 diff2_acf1   150    50     0.111 102.  0.912     0.912    ns
## 4 e_acf1       150    50    -1.71   85.4 0.0908    0.136    ns
## 5 entropy      150    50     1.30   94.2 0.196     0.220    ns
## 6 linearity    150    50    -1.31  113.  0.193     0.220    ns
## 7 spike        150    50     4.20  105.  0.0000555 0.000203 ***
## 8 trend        150    50    -4.17   94.3 0.0000677 0.000203 ***
## 9 x_acf1       150    50    -4.46   97.5 0.0000217 0.000195 ***
```

Metrics by Method

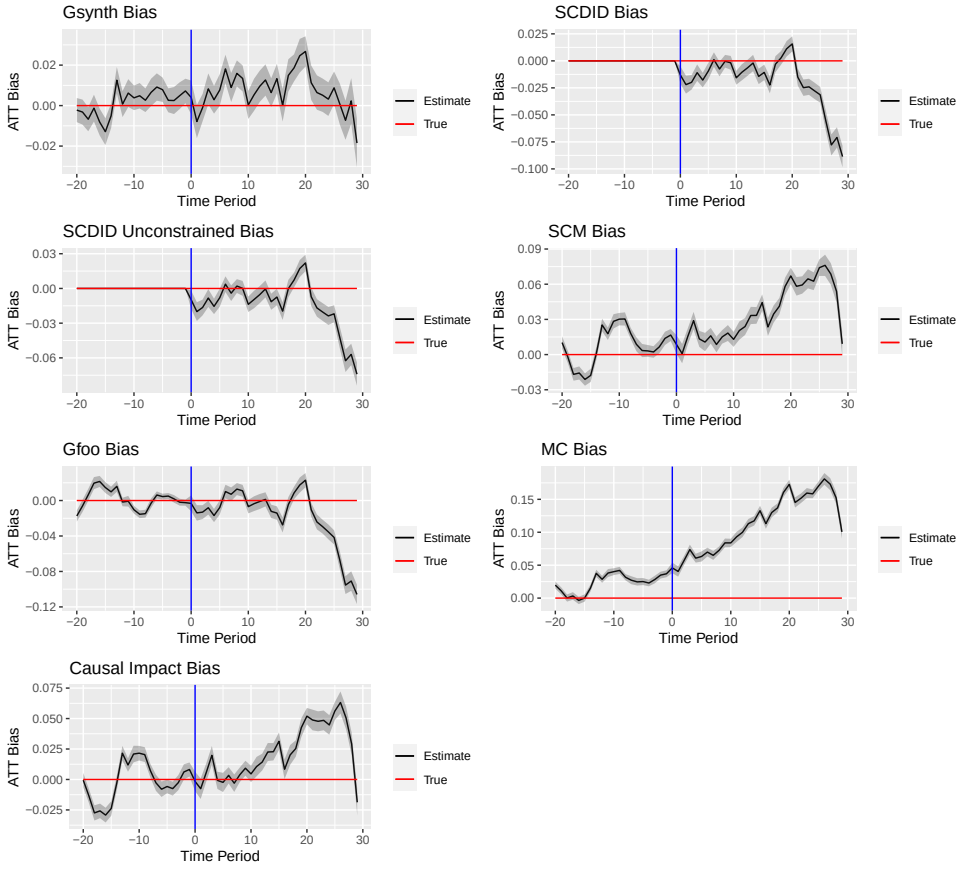
ab_no_het_loading_shift

Method	gsynth	scdid	scdid_uncon	scm	gfoo	mc	causalimpact
coverage							

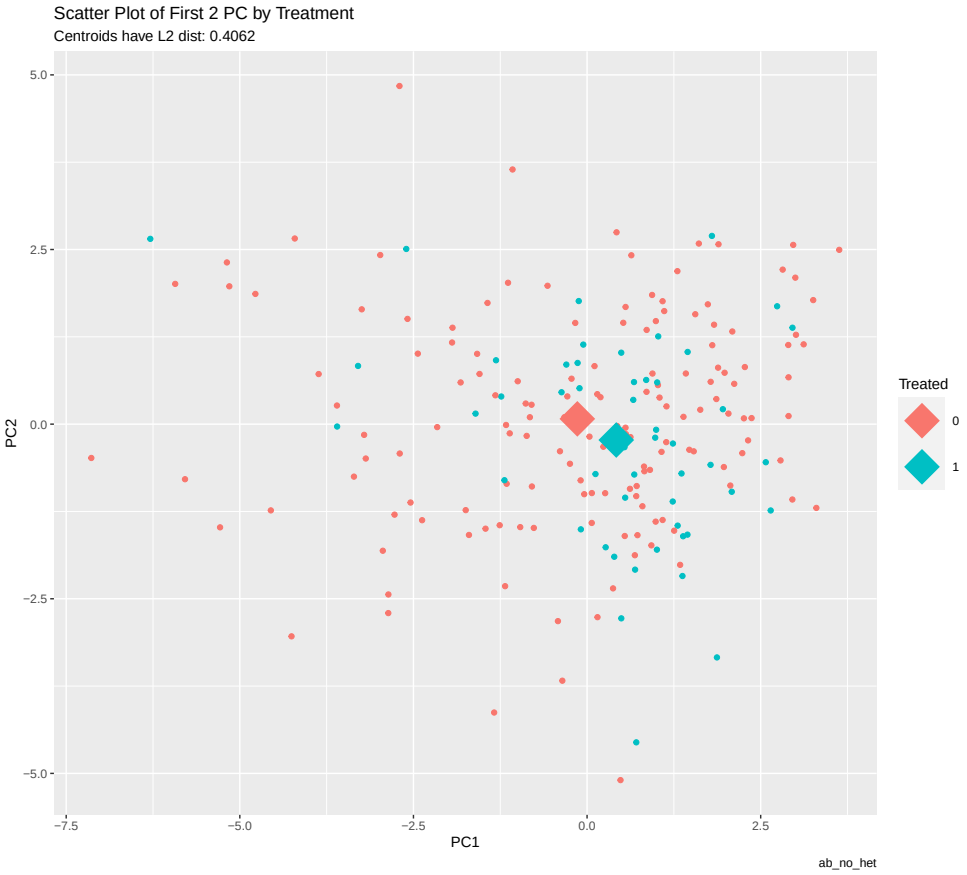
0	0.907	0.987	0.960	0.893	0.880	0.747	0.880
1	0.960	1.000	0.973	0.960	0.773	0.880	0.827
2	0.920	0.947	0.853	0.960	0.813	0.947	0.680
3	0.947	0.920	0.867	0.973	0.840	1.000	0.573
4	0.920	0.933	0.867	0.920	0.867	1.000	0.613
rmse							
0	0.219	0.275	0.242	0.235	0.270	0.471	0.240
1	0.225	0.287	0.250	0.241	0.294	0.509	0.247
2	0.225	0.292	0.251	0.241	0.310	0.540	0.256
3	0.224	0.283	0.244	0.240	0.324	0.533	0.251
4	0.225	0.278	0.245	0.238	0.340	0.521	0.255
bias							
0	−0.010	−0.004	−0.012	0.021	−0.038	0.108	−0.020
1	−0.012	−0.014	−0.021	0.007	−0.046	0.103	−0.031
2	−0.016	−0.027	−0.032	−0.010	−0.057	0.090	−0.053
3	−0.013	−0.034	−0.035	−0.017	−0.054	0.074	−0.059
4	−0.012	−0.037	−0.036	−0.022	−0.051	0.065	−0.061

Notes:

```
## [1] "ab_no_het"
```




```
## `summarise()` ungrouping output (override with `.groups` argument)
```




```
## # A tibble: 9 x 8
##   vars          n1    n2 statistic    df      p p.adj p.adj.signif
##   <chr>      <int> <int>     <dbl> <dbl>  <dbl> <dbl> <chr>
## 1 curvature    150    50   -0.492   89.3 0.624  0.702 ns
## 2 diff1_acf1   150    50   -0.505  101.  0.614  0.702 ns
## 3 diff2_acf1   150    50   -1.16    91.2 0.247  0.370 ns
## 4 e_acf1       150    50   -0.0119 107.  0.991  0.991 ns
## 5 entropy      150    50   -1.75   106.  0.0835 0.162 ns
## 6 linearity    150    50    2.05    83.4 0.043  0.162 ns
## 7 spike        150    50   -1.77    94.8 0.0797 0.162 ns
## 8 trend        150    50    2.07    94.7 0.0408 0.162 ns
## 9 x_acf1       150    50    1.71   102.  0.09   0.162 ns
```

Metrics by Method

ab_no_het

Method	gsynth	scdid	scdid_uncon	scm	gfoo	mc	causalimpact
coverage							

0	0.893	0.920	0.920	0.920	0.880	0.800	0.907
1	0.920	0.880	0.880	0.933	0.933	0.813	0.920
2	0.960	0.947	0.960	0.933	0.893	0.693	0.947
3	0.933	0.933	0.933	0.867	0.973	0.520	0.893
4	0.987	0.960	0.973	0.973	0.947	0.733	0.960
rmse							
0	0.217	0.233	0.231	0.224	0.231	0.263	0.227
1	0.222	0.236	0.233	0.226	0.237	0.269	0.227
2	0.220	0.236	0.233	0.226	0.233	0.275	0.226
3	0.221	0.238	0.236	0.229	0.239	0.278	0.228
4	0.224	0.245	0.242	0.231	0.245	0.288	0.232
bias							
0	0.004	−0.013	−0.010	0.009	−0.003	0.046	−0.002
1	−0.008	−0.022	−0.020	0.001	−0.014	0.040	−0.008
2	−0.001	−0.020	−0.016	0.016	−0.013	0.057	0.006
3	0.008	−0.011	−0.008	0.029	−0.008	0.074	0.020
4	0.003	−0.018	−0.016	0.013	−0.017	0.061	−0.001

Notes:

DGP Highlight 1: High ACF, Intercept Selection

Our first comparison involves a placebo DGP with high autocorrelation in the outcome process, and varying amounts of selection, as demonstrated in the R code below.[18] We set the total number of time periods to be 121 months, and include a total number of 200 entries, 80 of whom are treated. The key difference is in the value of "intercept_scale" – there are no shifts to the mean of the intercept distribution in the first process, but the mean is shifted down by 0.75 for control units in the second. Note that in setting the same seed for the DGPs (both in terms of $Y*$ and $\{\epsilon_m\}$), we home in on the impact of shifting said intercept.[19]

```
AA_data_no_sel_base=factor_synthetic_dgp(date_start="2010-01-01",
                                         first_treat="2017-07-01",
                                         date_end="2020-01-01",
                                         num_entries=200,
                                         prop_treated=0.25,
                                         treat_impact_sd = 0,
                                         treat_impact_mean = 0,
                                         rho=0.9,
                                         rescale_y_mean = 2.5e3,
                                         cov_overlap_scale = 0,
                                         seed=42)

AA_data_no_sel_unformatted=furrr::future_map(.x=seeds,
                                             .f=~noisify_draw(
                                               data_inp=AA_data_no_sel_base,
                                               seed=.x))


AA_data_sel_base=factor_synthetic_dgp(date_start="2010-01-01",
                                      first_treat="2017-07-01",
                                      date_end="2020-01-01",
                                      num_entries=200,
                                      prop_treated=0.25,
                                      treat_impact_sd = 0,
                                      treat_impact_mean = 0,
                                      rho=0.9,
                                      rescale_y_mean = 2.5e3,
                                      cov_overlap_scale = 0,
                                      intercept_scale = 0.75,
                                      seed=42)

AA_data_sel_unformatted=furrr::future_map(.x=seeds,
                                          .f=~noisify_draw(
                                            data_inp=AA_data_sel_base,
                                            seed=.x))
```

After estimating the Gsynth (IFE), SCDID, MC, and CausalImpact methods on the above data, we report the RMSE, bias, and coverage for each of the estimators below (Figure @ref(fig:high-acf-bias)). Without selection, we see that the 95% confidence intervals contain the true ATT roughly 85% of the time across the methods, with MC having the most stable performance despite under-covering. MC and CausalImpact alternate on having the lowest RMSE, though the differences are quite small, while MC consistently has the lowest bias. Selection seems to have a relevant effect only when considering the bias of the estimators, increasing it by as much as 10%. However, it does not effect the relative rankings among the estimators, as

[18] To be clear, we assign treatment randomly across the units in both processes, but in our selection condition here, we shift the mean of the intercept α_i down for control units compared to treatment units.

[19] We also rescale the output so that it is roughly lognormal around a mean of 9.

MC proves more robust to selection in this DGP. Because the plots are so similar regardless of selection, we only display the bias for the case with selection, though the table contains both sets of data.

Metrics by Method

Rho=0.9

	No				Yes			
Selection	gsynth	scdid	mc	causalimp	gsynth	scdid	mc	causalimp
coverage								
0	0.840	0.860	0.840	0.820	0.820	0.840	0.840	0.800
1	0.840	0.860	0.880	0.860	0.840	0.860	0.880	0.860
2	0.800	0.780	0.840	0.800	0.800	0.780	0.840	0.780
3	0.780	0.820	0.840	0.860	0.780	0.820	0.820	0.860
4	0.900	0.900	0.900	0.880	0.880	0.900	0.900	0.860
rmse								
0	1.050	1.088	1.047	1.046	1.051	1.088	1.048	1.047
1	1.012	1.057	1.009	1.012	1.013	1.058	1.010	1.013
2	1.044	1.095	1.042	1.036	1.047	1.097	1.045	1.039
3	1.049	1.091	1.048	1.046	1.051	1.094	1.049	1.048
4	0.992	1.051	0.992	0.999	0.994	1.054	0.994	1.001
bias								
0	0.133	0.133	0.111	0.148	0.140	0.145	0.117	0.154
1	0.138	0.150	0.121	0.132	0.146	0.159	0.127	0.139
2	0.176	0.172	0.165	0.179	0.184	0.183	0.171	0.186
3	0.164	0.168	0.157	0.154	0.172	0.174	0.164	0.160
4	0.118	0.125	0.114	0.129	0.126	0.133	0.121	0.135

Notes:

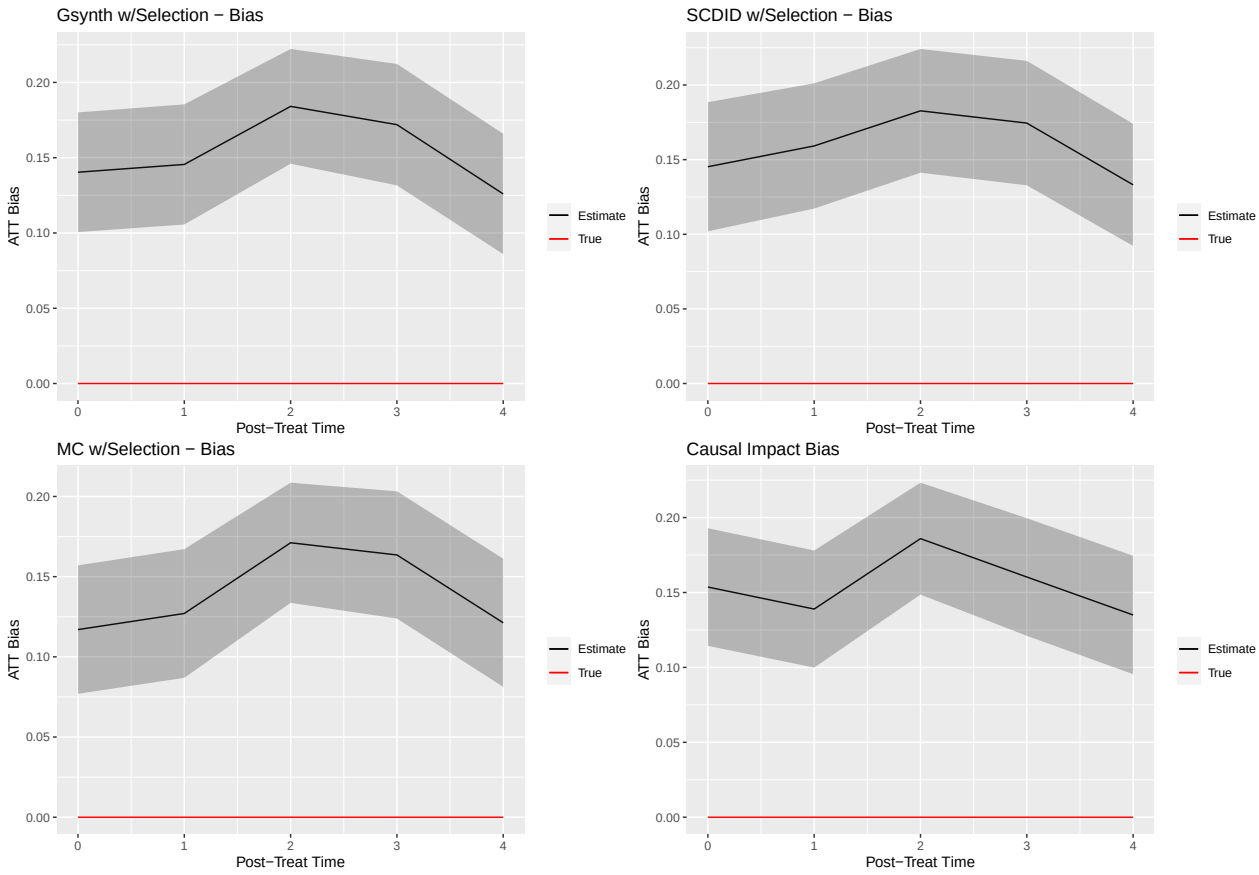


Figure 4: Bias Estimates