

Package ‘EventStudy’

October 31, 2026

Type Package

Title Event Study Analysis

Version 0.70

Date 2026-10-05

Maintainer Simon Mueller <sm@data-zoo.de>

Description Financial event study analysis. Implements the classical event study methodology of MacKinlay (1997) <<https://www.jstor.org/stable/2729691>> with multiple return models: market model, market adjusted, comparison period mean adjusted, Fama and French three-factor (1993) <[doi:10.1016/0304-405X\(93\)90023-5](https://doi.org/10.1016/0304-405X(93)90023-5)> and five-factor (2015) <[doi:10.1016/j.jfineco.2014.10.010](https://doi.org/10.1016/j.jfineco.2014.10.010)>, Carhart (1997) four-factor <[doi:10.1111/j.1540-6261.1997.tb03808.x](https://doi.org/10.1111/j.1540-6261.1997.tb03808.x)>, generalized autoregressive conditional heteroskedasticity (GARCH; Bollerslev, 1986) <[doi:10.1016/0304-4076\(86\)90063-1](https://doi.org/10.1016/0304-4076(86)90063-1)>, rolling window, and dynamic conditional correlation GARCH (DCC-GARCH; Engle, 2002) <[doi:10.1198/073500102288618487](https://doi.org/10.1198/073500102288618487)>. Provides parametric and non-parametric test statistics, including the standardized cross-sectional test of Boehmer, Musumeci and Poulsen (1991) <[doi:10.1016/0304-405X\(91\)90032-F](https://doi.org/10.1016/0304-405X(91)90032-F)> and its cross-correlation adjustment by Kolarik and Pynnönen (2010) <[doi:10.1093/rfs/hhq072](https://doi.org/10.1093/rfs/hhq072)>; buy-and-hold abnormal returns (Barber and Lyon, 1997) <[doi:10.1016/S0304-405X\(96\)00890-2](https://doi.org/10.1016/S0304-405X(96)00890-2)>; volume and volatility event studies; cross-sectional regression; intraday event studies; panel difference-in-differences event studies with the estimators of Callaway and Sant'Anna (2021) <[doi:10.1016/j.jeconom.2020.12.001](https://doi.org/10.1016/j.jeconom.2020.12.001)>, de Chaisemartin and D'Haultfoeuille (2020) <[doi:10.1257/aer.20181169](https://doi.org/10.1257/aer.20181169)>, and Borusyak, Jaravel and Spiess (2024) <[doi:10.1093/restud/rdae007](https://doi.org/10.1093/restud/rdae007)>; synthetic control methods (Abadie, Diamond and Hainmueller, 2010) <[doi:10.1198/jasa.2009.ap08746](https://doi.org/10.1198/jasa.2009.ap08746)>; wild bootstrap inference; multiple testing corrections; power analysis via simulation; data download helpers; automated reporting; and visualization tools. Results can be exported to comma-separated values (CSV), 'Excel', or 'LaTeX'.

URL <https://github.com/sipemu/eventstudy>,
<https://sipemu.github.io/eventstudy/>

BugReports <https://github.com/sipemu/eventstudy/issues>

License AGPL-3

Depends R (>= 4.1.0)

Imports R6, distributional, methods, plotly, tibble, ggplot2, dplyr,
purrr, tidyr, stringr, rlang, stats, tools, utils

Suggests testthat (>= 3.0.0), withr, lifecycle, generics, gridExtra,
knitr, rmarkdown, openxlsx, rugarch, sandwich, covr, rmgarch,
quadprog, callr, did, DIDmultiplegt, didimputation, tidyquant,
quantmod, zoo, httr2 (>= 1.0.0), jsonlite, tinytex, tinytable,
ragg

Encoding UTF-8

Config/testthat/edition 3

VignetteBuilder knitr

Config/roxygen2/version 8.0.0

LazyData true

NeedsCompilation no

Author Simon Mueller [cre, aut]

Repository CRAN

Date/Publication 2026-10-31 00:00:02 UTC

Contents

adjust_p_values	4
AnthropicProvider	5
ARTTest	7
BHARMModel	8
BHARTTest	9
BMPTTest	10
bootstrap_test	11
calculate_statistics	12
CalendarTimePortfolioTest	13
Carhart4FactorModel	14
CARTTest	15
car_by_group	16
car_quantiles	17
ComparisonPeriodMeanAdjustedModel	18
cross_sectional_regression	19
CSectTTest	20
CustomProvider	21
DCCGARCHModel	22
degenerate-input-contract	24
dieselgate	25
download_factor_data	26
download_risk_free_rate	27

download_stock_data	28
earnings_surprises	28
estimate_panel_event_study	30
estimate_synthetic_control	31
es_advise	32
es_colours	34
es_diagnostics	34
es_kb	35
es_report	36
eventstudy-deprecation	38
eventstudy-shape-contracts	39
EventStudyTask	39
export_results	42
FamaFrench3FactorModel	43
FamaFrench5FactorModel	44
fit_model	45
flag_robustness	46
format.Advice	47
format.es_advise	48
format.es_cross_sectional	48
format.es_diagnostics	49
format.es_simulation	49
format.EventStudySummary	50
GARCHModel	50
GeneralizedSignTest	51
generate_report	52
IntradayEventStudyTask	54
KolariPynnonenTest	56
LinearFactorModel	57
LogReturn	58
MarketAdjustedModel	59
MarketModel	61
ModelBase	62
model_diagnostics	64
MultiEventStatisticsSet	65
nonparametric_intraday_test	66
OpenAICompatProvider	67
PanelEventStudyTask	69
ParameterSet	71
PatellZTest	72
plot_car_distribution	73
plot_diagnostics	74
plot_event_study	74
plot_panel_event_study	75
plot_stocks	76
plot_synthetic_control	77
prepare_event_study	78
prepare_intraday_event_study	78

pretrend_test	79
print.Advice	80
print.es_advice	80
print.es_cross_sectional	81
print.es_diagnostics	81
print.EventStudySummary	82
provider	82
ProviderBase	83
RankTest	85
recommend_stat	86
report_table	87
ReturnCalculation	88
RollingWindowModel	89
run_event_study	90
sc_placebo_test	91
SignTest	92
SimpleReturn	93
simulate_event_study	94
SingleEventStatisticsSet	95
StatisticsSetBase	96
SyntheticControlTask	97
TestStatisticBase	98
theme_eventstudy	99
tidy.EventStudyTask	100
validate_task	101
VolatilityModel	102
VolumeModel	103
Index	105

adjust_p_values	<i>Adjust P-Values for Multiple Testing</i>
-----------------	---

Description

Computes adjusted p-values for AAR and CAAR test statistics across the event window, correcting for the multiple comparisons problem. Supports all methods available in [p.adjust](#).

Usage

```
adjust_p_values(task, method = "BH", stat_name = "CSectT", group = NULL)
```

Arguments

task	A fitted EventStudyTask with aar_caar_tbl populated.
method	Adjustment method passed to p.adjust . Common choices: "BH" (Benjamini-Hochberg, default), "bonferroni", "holm", "hochberg", "BY", "none".
stat_name	Name of the multi-event test statistic to adjust. Must match a column name in task\$aar_caar_tbl. Default "CSectT".
group	Optional group name to filter. If NULL, adjusts all groups.

Value

A tibble with columns from the original test statistic result plus p_raw_aar, p_adj_aar, p_raw_caar, p_adj_caar.

See Also

Other eventstudy-statistics: [ARTTest](#), [BHARTTest](#), [BMPTTest](#), [CARTTest](#), [CSectTTest](#), [CalendarTimePortfolioTest](#), [GeneralizedSignTest](#), [KolariPynnonenTest](#), [MultiEventStatisticsSet](#), [PatellZTest](#), [RankTest](#), [SignTest](#), [SingleEventStatisticsSet](#), [StatisticsSetBase](#), [TestStatisticBase](#), [bootstrap_test\(\)](#), [car_by_group\(\)](#), [car_quantiles\(\)](#), [cross_sectional_regression\(\)](#), [model_diagnostics\(\)](#), [pretrend_test\(\)](#), [simulate_event_study\(\)](#), [validate_task\(\)](#)

AnthropicProvider

AnthropicProvider

Description

A provider that issues POST {base_url}/v1/messages against the native Anthropic Messages API. The request carries the mandatory max_tokens field and the anthropic-version header; when a schema is supplied it adds a tool-use input_schema block (tools + tool_choice) to obtain structured output, and the response extractor prefers that tool call's structured input while falling back to the first text block – so the provider never crashes when the model replies with plain text instead of a tool call.

The API key is resolved at CALL time from ANTHROPIC_API_KEY (never at construction, never stored on the object) and attached with req_headers_redacted("x-api-key" = key), which redacts the key in every print/error path. Note req_auth_bearer_token does NOT redact an x-api-key header – the redacted-headers helper is mandatory here. A missing key, transport/timeout failure, non-2xx status, malformed body, or missing completion text each degrades to exactly one warning() plus an 'es_provider_response' with text = NA_character_ – it never crashes the session and never returns a fabricated completion.

httr2 is required only for this provider and is guarded by requireNamespace() at the top of complete(), so the package installs and R CMD checks cleanly with httr2 absent.

Super class

[ProviderBase](#) -> AnthropicProvider

Public fields

`max_tokens` Integer max completion tokens (Anthropic **REQUIRES** this).

Methods**Public methods:**

- [AnthropicProvider\\$new\(\)](#)
- [AnthropicProvider\\$complete\(\)](#)
- [AnthropicProvider\\$clone\(\)](#)

AnthropicProvider\$new(): Construct an AnthropicProvider. Stores non-secret config only; the API key is resolved at call time inside `complete()`.

Usage:

```
AnthropicProvider$new(
  model,
  base_url = "https://api.anthropic.com",
  max_tokens = 1024L,
  ...
)
```

Arguments:

`model` Character model identifier, e.g. "claude-opus-4-8".

`base_url` Character base URL. Defaults to the Anthropic cloud endpoint.

`max_tokens` Integer maximum number of tokens to generate. The Anthropic Messages API **REQUIRES** this field; defaults to 1024.

`...` Ignored; accepted for forward-compatibility.

AnthropicProvider\$complete(): Complete a prompt via POST {`base_url`}/v1/messages. Resolves the key at call time; on a missing key or any HTTP/parse failure returns one warning + NA. Structured output is **OPTIONAL**: when schema is supplied a tool-use input_schema block is added and the extractor prefers the tool call's input, falling back to the first text block.

Usage:

```
AnthropicProvider$complete(prompt, schema = NULL, ...)
```

Arguments:

`prompt` Character prompt.

`schema` Optional structured-output schema (list) or NULL.

`...` Ignored; accepted for forward-compatibility.

Returns: An 'es_provider_response' (see `ProviderBase$complete`).

AnthropicProvider\$clone(): The objects of this class are cloneable with this method.

Usage:

```
AnthropicProvider$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-advisor: `CustomProvider`, `OpenAICompatProvider`, `ProviderBase`, `es_advise()`, `es_diagnostics()`, `es_kb()`, `flag_robustness()`, `provider()`, `recommend_stat()`

Examples

```
## Not run:
# Reads ANTHROPIC_API_KEY from the environment at call time:
p <- AnthropicProvider$new(model = "claude-opus-4-8")
p$complete("Summarise these event-study diagnostics")

# Structured output via a tool-use input_schema:
schema <- list(type = "object",
               properties = list(advice = list(type = "string")))
p$complete("Recommend a test statistic", schema = schema)

## End(Not run)
```

ARTTest

Abnormal Return T Statistic (ART)

Description

The AR t-test is a statistical method used to determine whether the abnormal return of a security on a specific day is significantly different from zero. This test helps researchers identify whether the event of interest has a significant impact on the security's return at a particular point in time.

See also https://eventstudy.de/statistics/ar_car_statistics.html

Super class

`TestStatisticBase` -> ARTTest

Public fields

`name` Short code of the test statistic.

Methods**Public methods:**

- `ARTTest$compute()`
- `ARTTest$clone()`

`ARTTest$compute()`: Computes the test AR test statistics for a single event.

Usage:

`ARTTest$compute(data_tbl, model)`

Arguments:

`data_tbl` The data for a single event with calculated abnormal returns.

`model` The fitted model that includes the necessary information for calculating the test statistic.

`ARTTest$clone()`: The objects of this class are cloneable with this method.

Usage:

`ARTTest$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-statistics: [BHARTTest](#), [BMPTTest](#), [CARTTest](#), [CSectTTest](#), [CalendarTimePortfolioTest](#), [GeneralizedSignTest](#), [KolariPynnonenTest](#), [MultiEventStatisticsSet](#), [PatellZTest](#), [RankTest](#), [SignTest](#), [SingleEventStatisticsSet](#), [StatisticsSetBase](#), [TestStatisticBase](#), [adjust_p_values\(\)](#), [bootstrap_test\(\)](#), [car_by_group\(\)](#), [car_quantiles\(\)](#), [cross_sectional_regression\(\)](#), [model_diagnostics\(\)](#), [pretrend_test\(\)](#), [simulate_event_study\(\)](#), [validate_task\(\)](#)

BHARModel

Buy-and-Hold Abnormal Returns (BHAR) Model

Description

Implements Buy-and-Hold Abnormal Returns for long-horizon event studies. BHAR compounds returns over the event window instead of summing:

$$BHAR_i = \prod (1 + R_{i,t}) - \prod (1 + R_{benchmark,t})$$

The benchmark is the market/index return by default. This model is appropriate for long-horizon studies (months/years) where compounding effects matter.

Super class

[ModelBase](#) -> BHARModel

Public fields

`model_name` Name of the model.

Methods

Public methods:

- [BHARModel\\$fit\(\)](#)
- [BHARModel\\$abnormal_returns\(\)](#)
- [BHARModel\\$clone\(\)](#)

`BHARModel$fit()`: Fit the BHAR model. Computes estimation window statistics.

Usage:

```
BHARModel$fit(data_tbl)
```

Arguments:

`data_tbl` Data frame or tibble.

`BHARModel$abnormal_returns()`: Calculate abnormal returns using buy-and-hold compounding.

Usage:

```
BHARModel$abnormal_returns(data_tbl)
```

Arguments:

`data_tbl` Data frame or tibble.

`BHARModel$clone()`: The objects of this class are cloneable with this method.

Usage:

```
BHARModel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-models: [Carhart4FactorModel](#), [ComparisonPeriodMeanAdjustedModel](#), [DCCGARCHModel](#), [FamaFrench3FactorModel](#), [FamaFrench5FactorModel](#), [GARCHModel](#), [LinearFactorModel](#), [LogReturn](#), [MarketAdjustedModel](#), [MarketModel](#), [ModelBase](#), [ReturnCalculation](#), [RollingWindowModel](#), [SimpleReturn](#), [VolatilityModel](#), [VolumeModel](#)

BHARTTest

Buy-and-Hold Abnormal Return T Test (BHARTTest)

Description

Tests whether the BHAR for a single event is significantly different from zero. The BHAR is the difference between compounded firm returns and compounded benchmark returns over the event window.

Super class

[TestStatisticBase](#) -> BHARTTest

Public fields

`name` Short code of the test statistic.

Methods

Public methods:

- [BHARTTest\\$compute\(\)](#)
- [BHARTTest\\$clone\(\)](#)

[BHARTTest\\$compute\(\)](#): Computes the BHAR t test for a single event.

Usage:

```
BHARTTest$compute(data_tbl, model)
```

Arguments:

`data_tbl` The data for a single event with calculated abnormal returns.

`model` The fitted model.

[BHARTTest\\$clone\(\)](#): The objects of this class are cloneable with this method.

Usage:

```
BHARTTest$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-statistics: [ARTTest](#), [BMPTest](#), [CARTTest](#), [CSectTTest](#), [CalendarTimePortfolioTest](#), [GeneralizedSignTest](#), [KolariPynnonenTest](#), [MultiEventStatisticsSet](#), [PatellZTest](#), [RankTest](#), [SignTest](#), [SingleEventStatisticsSet](#), [StatisticsSetBase](#), [TestStatisticBase](#), [adjust_p_values\(\)](#), [bootstrap_test\(\)](#), [car_by_group\(\)](#), [car_quantiles\(\)](#), [cross_sectional_regression\(\)](#), [model_diagnostics\(\)](#), [pretrend_test\(\)](#), [simulate_event_study\(\)](#), [validate_task\(\)](#)

BMPTest

BMP Test (Boehmer, Musumeci, Poulsen 1991)

Description

Standardized cross-sectional test that is robust to event-induced variance increases. The BMP test standardizes abnormal returns by each event's MODEL sigma (the estimation-window residual standard deviation, i.e. `model$statistics$sigma`) – NOT by the forecast-error-corrected sigma – then applies a cross-sectional t-test to these standardized residuals.

Super class

[TestStatisticBase](#) -> BMPTest

Public fields

`name` Short code of the test statistic.

Methods

Public methods:

- [BMPTest\\$compute\(\)](#)
- [BMPTest\\$clone\(\)](#)

BMPTest\$compute(): Computes the BMP standardized cross-sectional test.

Usage:

```
BMPTest$compute(data_tbl, model)
```

Arguments:

`data_tbl` The data for a multiple event with calculated abnormal returns.

`model` The fitted model containing sigma estimates.

BMPTest\$clone(): The objects of this class are cloneable with this method.

Usage:

```
BMPTest$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-statistics: [ARTTest](#), [BHARTTest](#), [CARTTest](#), [CSectTTest](#), [CalendarTimePortfolioTest](#), [GeneralizedSignTest](#), [KolariPynnonenTest](#), [MultiEventStatisticsSet](#), [PatellZTest](#), [RankTest](#), [SignTest](#), [SingleEventStatisticsSet](#), [StatisticsSetBase](#), [TestStatisticBase](#), [adjust_p_values\(\)](#), [bootstrap_test\(\)](#), [car_by_group\(\)](#), [car_quantiles\(\)](#), [cross_sectional_regression\(\)](#), [model_diagnostics\(\)](#), [pretrend_test\(\)](#), [simulate_event_study\(\)](#), [validate_task\(\)](#)

bootstrap_test

Wild Bootstrap Inference for Event Studies

Description

Computes bootstrap p-values for AAR and CAAR test statistics using the wild bootstrap approach. Per-firm random weights preserve the cross-sectional dependence structure while randomizing the sign of abnormal returns under the null hypothesis.

Usage

```
bootstrap_test(
  task,
  n_boot = 999L,
  weight_type = "rademacher",
  statistic = "both",
  group = NULL,
  seed = NULL
)
```

Arguments

task	A fitted EventStudyTask with abnormal returns computed.
n_boot	Number of bootstrap replications. Default 999.
weight_type	Type of bootstrap weights: "rademacher" (default, +1/-1 with equal probability) or "mammen" (Mammen two-point distribution).
statistic	Which statistic to bootstrap: "aar", "caar", or "both" (default).
group	Optional group name to filter.
seed	Optional seed for reproducibility.

Value

A tibble with columns: relative_index, observed_aar, observed_caar, boot_p_aar, boot_p_caar.

See Also

Other eventstudy-statistics: [ARTTest](#), [BHARTTest](#), [BMPTTest](#), [CARTTest](#), [CSectTTest](#), [CalendarTimePortfolioTest](#), [GeneralizedSignTest](#), [KolariPynnonenTest](#), [MultiEventStatisticsSet](#), [PatellZTest](#), [RankTest](#), [SignTest](#), [SingleEventStatisticsSet](#), [StatisticsSetBase](#), [TestStatisticBase](#), [adjust_p_values\(\)](#), [car_by_group\(\)](#), [car_quantiles\(\)](#), [cross_sectional_regression\(\)](#), [model_diagnostics\(\)](#), [pretrend_test\(\)](#), [simulate_event_study\(\)](#), [validate_task\(\)](#)

calculate_statistics	<i>Calculate test statistics</i>
----------------------	----------------------------------

Description

Calculate test statistics

Usage

```
calculate_statistics(task, parameter_set)
```

Arguments

task	The event study task
parameter_set	The parameter set that defines the event study.

Value

task

See Also

[run_event_study](#), [prepare_event_study](#), [fit_model](#)

Other eventstudy-pipeline: [EventStudyTask](#), [ParameterSet](#), [fit_model\(\)](#), [prepare_event_study\(\)](#), [run_event_study\(\)](#)

CalendarTimePortfolioTest

Calendar-Time Portfolio Test

Description

Aggregates event-firm returns into calendar-time portfolios and tests whether the portfolio intercept (alpha) is significantly different from zero, using the Brown and Warner (1980, 1985) crude-dependence-adjustment: the standard deviation of the ESTIMATION-window cross-event average abnormal return (AAR) series is used as the (calendar-)time-series standard deviation, so the denominator is independent of the event window and of any event-window shock magnitude.

For each relative event day, the test forms an equal-weighted portfolio of all event firms' abnormal returns and computes $caltime_t = AAR_t / sd(AAR_{estimation})$, with degrees of freedom $n_{estimation_days} - 1$ (the number of finite estimation-window AAR observations), exposed as `attr(result, "caltime_df")`.

Super class

`TestStatisticBase` -> `CalendarTimePortfolioTest`

Public fields

`name` Short code of the test statistic.

Methods

Public methods:

- `CalendarTimePortfolioTest$compute()`
- `CalendarTimePortfolioTest$clone()`

`CalendarTimePortfolioTest$compute()`: Computes the calendar-time portfolio test.

Usage:

```
CalendarTimePortfolioTest$compute(data_tbl, model)
```

Arguments:

`data_tbl` The data for multiple events with calculated abnormal returns.

`model` The fitted models (unused directly).

`CalendarTimePortfolioTest$clone()`: The objects of this class are cloneable with this method.

Usage:

```
CalendarTimePortfolioTest$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

References

Brown, S. J. and Warner, J. B. (1980). Measuring security price performance. *Journal of Financial Economics*, 8(3), 205–258.

Brown, S. J. and Warner, J. B. (1985). Using daily stock returns: The case of event studies. *Journal of Financial Economics*, 14(1), 3–31.

See Also

Other eventstudy-statistics: [ARTTest](#), [BHARTTest](#), [BMPTTest](#), [CARTTest](#), [CSectTTest](#), [GeneralizedSignTest](#), [KolariPynnonenTest](#), [MultiEventStatisticsSet](#), [PatellZTest](#), [RankTest](#), [SignTest](#), [SingleEventStatisticsSet](#), [StatisticsSetBase](#), [TestStatisticBase](#), [adjust_p_values\(\)](#), [bootstrap_test\(\)](#), [car_by_group\(\)](#), [car_quantiles\(\)](#), [cross_sectional_regression\(\)](#), [model_diagnostics\(\)](#), [pretrend_test\(\)](#), [simulate_event_study\(\)](#), [validate_task\(\)](#)

Carhart4FactorModel	<i>Carhart Four-Factor Model</i>
---------------------	----------------------------------

Description

Implements the Carhart (1997) four-factor model, which extends the Fama-French three-factor model with a momentum factor:

$$R_i - R_f = \alpha + \beta_m(R_m - R_f) + \beta_s SMB + \beta_h HML + \beta_{mom} MOM + \epsilon$$

Requires columns: excess_return, market_excess, smb, hml, mom.

Super classes

[ModelBase](#) -> [LinearFactorModel](#) -> Carhart4FactorModel

Public fields

model_name Name of the model.

formula The four-factor regression formula.

required_columns Required data columns.

Methods

Public methods:

- [Carhart4FactorModel\\$new\(\)](#)
- [Carhart4FactorModel\\$abnormal_returns\(\)](#)
- [Carhart4FactorModel\\$clone\(\)](#)

[Carhart4FactorModel\\$new\(\)](#): Create a new Carhart4FactorModel.

Usage:

```
Carhart4FactorModel$new(use_hac = FALSE, hac_lag = NULL)
```

Arguments:

use_hac Logical. Use HAC (Newey-West) standard errors.

hac_lag Integer or NULL. Lag truncation for Newey-West.

Carhart4FactorModel\$abnormal_returns(): Calculate abnormal returns using the four-factor model.

Usage:

```
Carhart4FactorModel$abnormal_returns(data_tbl)
```

Arguments:

data_tbl Data frame or tibble.

Carhart4FactorModel\$clone(): The objects of this class are cloneable with this method.

Usage:

```
Carhart4FactorModel$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other eventstudy-models: [BHARModel](#), [ComparisonPeriodMeanAdjustedModel](#), [DCCGARCHModel](#), [FamaFrench3FactorModel](#), [FamaFrench5FactorModel](#), [GARCHModel](#), [LinearFactorModel](#), [LogReturn](#), [MarketAdjustedModel](#), [MarketModel](#), [ModelBase](#), [ReturnCalculation](#), [RollingWindowModel](#), [SimpleReturn](#), [VolatilityModel](#), [VolumeModel](#)

CARTTest

Cumulative Abnormal Return T Statistic (CART)

Description

The CAR t-test is a statistical method used to determine whether the cumulative abnormal return of a security over an event window is significantly different from zero. This test helps researchers identify whether the event of interest has a significant impact on the security's return over the entire event window, considering the cumulative effects of the event.

See also https://eventstudy.de/statistics/ar_car_statistics.html

Super class

[TestStatisticBase](#) -> CARTTest

Public fields

name Short code of the test statistic.

Methods

Public methods:

- [CARTTest\\$compute\(\)](#)
- [CARTTest\\$clone\(\)](#)

CARTTest\$compute(): Computes the test CAR test statistics for a single event.

Usage:

`CARTTest$compute(data_tbl, model)`

Arguments:

`data_tbl` The data for a single event with calculated abnormal returns.
`model` The fitted model that includes the necessary information for calculating the test statistic.

CARTTest\$clone(): The objects of this class are cloneable with this method.

Usage:

`CARTTest$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-statistics: [ARTTest](#), [BHARTTest](#), [BMPTTest](#), [CSectTTest](#), [CalendarTimePortfolioTest](#), [GeneralizedSignTest](#), [KolariPynnonenTest](#), [MultiEventStatisticsSet](#), [PatellZTest](#), [RankTest](#), [SignTest](#), [SingleEventStatisticsSet](#), [StatisticsSetBase](#), [TestStatisticBase](#), [adjust_p_values\(\)](#), [bootstrap_test\(\)](#), [car_by_group\(\)](#), [car_quantiles\(\)](#), [cross_sectional_regression\(\)](#), [model_diagnostics\(\)](#), [pretrend_test\(\)](#), [simulate_event_study\(\)](#), [validate_task\(\)](#)

car_by_group	<i>Compare CARs Across Groups</i>
--------------	-----------------------------------

Description

Performs a t-test (two groups) or ANOVA (3+ groups) to test whether CARs differ significantly across groups.

Usage

`car_by_group(task, group_var = "group", car_window = NULL)`

Arguments

<code>task</code>	A fitted EventStudyTask.
<code>group_var</code>	Name of the grouping variable. Defaults to "group" (the standard group column in EventStudyTask).
<code>car_window</code>	Optional two-element vector for CAR window.

Value

A list with test results and group-level summary statistics.

See Also

Other eventstudy-statistics: [ARTTest](#), [BHARTTest](#), [BMPTTest](#), [CARTTest](#), [CSectTTest](#), [CalendarTimePortfolioTest](#), [GeneralizedSignTest](#), [KolariPynnonenTest](#), [MultiEventStatisticsSet](#), [PatellZTest](#), [RankTest](#), [SignTest](#), [SingleEventStatisticsSet](#), [StatisticsSetBase](#), [TestStatisticBase](#), [adjust_p_values\(\)](#), [bootstrap_test\(\)](#), [car_quantiles\(\)](#), [cross_sectional_regression\(\)](#), [model_diagnostics\(\)](#), [pretrend_test\(\)](#), [simulate_event_study\(\)](#), [validate_task\(\)](#)

car_quantiles	<i>CAR Quantiles</i>
---------------	----------------------

Description

Compute quantiles of the CAR distribution across events.

Usage

```
car_quantiles(task, probs = c(0.05, 0.25, 0.5, 0.75, 0.95), car_window = NULL)
```

Arguments

task	A fitted EventStudyTask.
probs	Probability vector for quantiles. Default <code>c(0.05, 0.25, 0.5, 0.75, 0.95)</code> .
car_window	Optional two-element vector for CAR window.

Value

A named numeric vector of quantiles.

See Also

Other eventstudy-statistics: [ARTTest](#), [BHARTTest](#), [BMPTTest](#), [CARTTest](#), [CSectTTest](#), [CalendarTimePortfolioTest](#), [GeneralizedSignTest](#), [KolariPynnonenTest](#), [MultiEventStatisticsSet](#), [PatellZTest](#), [RankTest](#), [SignTest](#), [SingleEventStatisticsSet](#), [StatisticsSetBase](#), [TestStatisticBase](#), [adjust_p_values\(\)](#), [bootstrap_test\(\)](#), [car_by_group\(\)](#), [cross_sectional_regression\(\)](#), [model_diagnostics\(\)](#), [pretrend_test\(\)](#), [simulate_event_study\(\)](#), [validate_task\(\)](#)

ComparisonPeriodMeanAdjustedModel

Comparison Period Mean Adjusted Model

Description

The Comparison Period Mean Adjusted Model is another relatively simple approach used in event studies to estimate the expected returns of a stock and calculate its abnormal returns during an event window. This model is based on the assumption that a stock's expected return during the event window is equal to its average return during a comparison period (typically a pre-event period). This model is particularly useful when researchers want to control for a stock's historical performance and do not wish to rely on market return data.

Super class

`ModelBase` -> ComparisonPeriodMeanAdjustedModel

Public fields

`model_name` Name of the model.

Methods

Public methods:

- `ComparisonPeriodMeanAdjustedModel$fit()`
- `ComparisonPeriodMeanAdjustedModel$abnormal_returns()`
- `ComparisonPeriodMeanAdjustedModel$clone()`

`ComparisonPeriodMeanAdjustedModel$fit()`: Fit the model with given data.

Usage:

`ComparisonPeriodMeanAdjustedModel$fit(data_tbl)`

Arguments:

`data_tbl` Data frame or tibble containing the data to fit.

`ComparisonPeriodMeanAdjustedModel$abnormal_returns()`: Calculate the abnormal returns with given data.

Usage:

`ComparisonPeriodMeanAdjustedModel$abnormal_returns(data_tbl)`

Arguments:

`data_tbl` Data frame or tibble containing the data to calculate abnormal returns.

`ComparisonPeriodMeanAdjustedModel$clone()`: The objects of this class are cloneable with this method.

Usage:

`ComparisonPeriodMeanAdjustedModel$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-models: [BHARModel](#), [Carhart4FactorModel](#), [DCCGARCHModel](#), [FamaFrench3FactorModel](#), [FamaFrench5FactorModel](#), [GARCHModel](#), [LinearFactorModel](#), [LogReturn](#), [MarketAdjustedModel](#), [MarketModel](#), [ModelBase](#), [ReturnCalculation](#), [RollingWindowModel](#), [SimpleReturn](#), [VolatilityModel](#), [VolumeModel](#)

cross_sectional_regression

Cross-Sectional Regression of CARs

Description

Regress cumulative abnormal returns (CARs) on firm characteristics to explain cross-sectional variation in event effects. Supports OLS with heteroskedasticity-consistent (HC) standard errors.

Usage

```
cross_sectional_regression(
  task,
  formula,
  data,
  car_window = NULL,
  robust = TRUE
)
```

Arguments

task	A fitted EventStudyTask with abnormal returns computed.
formula	A formula with the response on the left (ignored; CAR is always the dependent variable) and explanatory variables on the right, e.g., <code>~ log_market_cap + leverage</code> .
data	A data frame of firm characteristics. Must contain an <code>event_id</code> column to merge with CARs.
car_window	Two-element integer vector specifying the CAR window as <code>c(start, end)</code> relative indices. Default is the full event window.
robust	Logical. If TRUE and the sandwich package is available, compute HC1 robust standard errors. Default TRUE.

Value

A list with class "es_cross_sectional" containing:

model The fitted `lm` object
coefficients Coefficient table with (robust) standard errors
r_squared R-squared of the regression
n_obs Number of observations
car_data The merged CAR + characteristics data

See Also

Other eventstudy-statistics: [ARTTest](#), [BHARTTest](#), [BMPTTest](#), [CARTTest](#), [CSectTTest](#), [CalendarTimePortfolioTest](#), [GeneralizedSignTest](#), [KolariPynnonenTest](#), [MultiEventStatisticsSet](#), [PatellZTest](#), [RankTest](#), [SignTest](#), [SingleEventStatisticsSet](#), [StatisticsSetBase](#), [TestStatisticBase](#), [adjust_p_values\(\)](#), [bootstrap_test\(\)](#), [car_by_group\(\)](#), [car_quantiles\(\)](#), [model_diagnostics\(\)](#), [pretrend_test\(\)](#), [simulate_event_study\(\)](#), [validate_task\(\)](#)

CSectTTest

*Cross-Sectional T Test (CSectTTest)***Description**

The Cross-Sectional Test (CSect T) is a statistical tool employed in event studies to evaluate the null hypothesis that the average abnormal return at the event date is zero. The test statistic is distributed as t_{N-1} , where N is the number of events in that group.

See also https://eventstudy.de/statistics/aar_caar_statistics.html

Super class

[TestStatisticBase](#) -> CSectTTest

Public fields

name Short code of the test statistic.

Methods**Public methods:**

- [CSectTTest\\$compute\(\)](#)
- [CSectTTest\\$clone\(\)](#)

[CSectTTest\\$compute\(\)](#): Computes the AAR/CAAR cross-sectional t test statistics.

Usage:

[CSectTTest\\$compute\(data_tbl, model\)](#)

Arguments:

data_tbl The data for a multiple event with calculated abnormal returns.

model The fitted model: not necessary for this test statistics.

[CSectTTest\\$clone\(\)](#): The objects of this class are cloneable with this method.

Usage:

[CSectTTest\\$clone\(deep = FALSE\)](#)

Arguments:

deep Whether to make a deep clone.

See Also

Other eventstudy-statistics: [ARTTest](#), [BHARTTest](#), [BMPTTest](#), [CARTTest](#), [CalendarTimePortfolioTest](#), [GeneralizedSignTest](#), [KolariPynnonenTest](#), [MultiEventStatisticsSet](#), [PatellZTest](#), [RankTest](#), [SignTest](#), [SingleEventStatisticsSet](#), [StatisticsSetBase](#), [TestStatisticBase](#), [adjust_p_values\(\)](#), [bootstrap_test\(\)](#), [car_by_group\(\)](#), [car_quantiles\(\)](#), [cross_sectional_regression\(\)](#), [model_diagnostics\(\)](#), [pretrend_test\(\)](#), [simulate_event_study\(\)](#), [validate_task\(\)](#)

CustomProvider

CustomProvider

Description

A provider that wraps a user-supplied function(prompt, schema, ...) -> list | character. This is the offline end-to-end seam and the escape hatch for backends the built-in HTTP providers do not cover. It uses NO network and needs neither http2 nor jsonlite.

The user function is run inside tryCatch: if it errors, the provider degrades to exactly one warning() plus an 'es_provider_response' with text = NA_character_ – it never crashes the session.

Super class

[ProviderBase](#) -> CustomProvider

Methods**Public methods:**

- [CustomProvider\\$new\(\)](#)
- [CustomProvider\\$complete\(\)](#)
- [CustomProvider\\$clone\(\)](#)

[CustomProvider\\$new\(\)](#): Construct a CustomProvider.

Usage:

```
CustomProvider$new(fn, model = NULL, ...)
```

Arguments:

fn A function of (prompt, schema, ...) returning a list or a character completion. Required.

model Optional character model identifier (informational only).

... Ignored; forwarded for forward-compatibility.

[CustomProvider\\$complete\(\)](#): Run the wrapped user function and wrap its result in an 'es_provider_response'. A throwing function degrades to one warning + NA.

Usage:

```
CustomProvider$complete(prompt, schema = NULL, ...)
```

Arguments:

prompt Character prompt passed to the user function.

schema Optional structured-output schema passed through to the user function.

... Additional arguments forwarded to the user function.

Returns: An 'es_provider_response' (see ProviderBase\$complete).

CustomProvider\$clone(): The objects of this class are cloneable with this method.

Usage:

```
CustomProvider$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other eventstudy-advisor: [AnthropicProvider](#), [OpenAICompatProvider](#), [ProviderBase](#), [es_advise\(\)](#), [es_diagnostics\(\)](#), [es_kb\(\)](#), [flag_robustness\(\)](#), [provider\(\)](#), [recommend_stat\(\)](#)

Examples

```
# In-process, no network: the user function supplies the completion text.
p <- CustomProvider$new(function(prompt, schema) "canned advice")
res <- p$complete("Summarise these diagnostics")
res$text
res$source
```

DCCGARCHModel

DCC-GARCH Model

Description

Event study model using Dynamic Conditional Correlation GARCH for time-varying beta estimation. Requires the **rmgarch** package. The bivariate DCC-GARCH model captures both time-varying volatility and time-varying correlation between firm and market returns, yielding a time-varying beta: $\beta_t = Cov(R_{firm}, R_{market})_t / Var(R_{market})_t$.

Super class

[ModelBase](#) -> DCCGARCHModel

Public fields

model_name Name of the model.

garch_order GARCH order for each univariate model. Default c(1,1).

dcc_order DCC order. Default c(1,1).

Methods

Public methods:

- [DCCGARCHModel\\$new\(\)](#)
- [DCCGARCHModel\\$fit\(\)](#)
- [DCCGARCHModel\\$abnormal_returns\(\)](#)
- [DCCGARCHModel\\$clone\(\)](#)

`DCCGARCHModel$new()`: Create a new DCCGARCHModel.

Usage:

```
DCCGARCHModel$new(garch_order = c(1, 1), dcc_order = c(1, 1))
```

Arguments:

`garch_order` GARCH(p,q) order for univariate models.

`dcc_order` DCC(a,b) order.

`DCCGARCHModel$fit()`: Fit the DCC-GARCH model on the estimation window.

Usage:

```
DCCGARCHModel$fit(data_tbl)
```

Arguments:

`data_tbl` Data frame or tibble with `firm_returns`, `index_returns`, `estimation_window`, `event_window` columns.

`DCCGARCHModel$abnormal_returns()`: Calculate abnormal returns using the last conditional beta.

Usage:

```
DCCGARCHModel$abnormal_returns(data_tbl)
```

Arguments:

`data_tbl` Data frame or tibble.

`DCCGARCHModel$clone()`: The objects of this class are cloneable with this method.

Usage:

```
DCCGARCHModel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-models: [BHARModel](#), [Carhart4FactorModel](#), [ComparisonPeriodMeanAdjustedModel](#), [FamaFrench3FactorModel](#), [FamaFrench5FactorModel](#), [GARCHModel](#), [LinearFactorModel](#), [LogReturn](#), [MarketAdjustedModel](#), [MarketModel](#), [ModelBase](#), [ReturnCalculation](#), [RollingWindowModel](#), [SimpleReturn](#), [VolatilityModel](#), [VolumeModel](#)

degenerate-input-contract

Degenerate-Input Contract for EventStudy Models

Description

The degenerate-input contract defines how all EventStudy return models behave when the estimation data is degenerate. Two modes are supported:

Lenient (default): The model sets `is_fitted = FALSE`, emits exactly one `warning()` per `(event_id, firm_symbol)` per fit call, and propagates NA through all abnormal returns and downstream statistics. No event is silently dropped; zeros are never substituted for NA.

Strict: The model raises a descriptive `stop()` error naming the component, `event_id`, and `firm_symbol`, plus the specific reason for the degeneracy.

Configuration:

- Via `ParameterSet`: `ParameterSet$new(degenerate_handling = "strict")`
- Via package option: `options(EventStudy.degenerate_handling = "strict")`

The `ParameterSet` field takes precedence over the package option; if neither is set, the default "lenient" mode is used.

Degenerate conditions covered:

- Fewer than 2 finite observations in the estimation window (insufficient observations for OLS).
- Zero or near-zero variance in index returns (`sd < .Machine$double.eps`), making OLS estimation undefined.
- Single-event group (relevant for multi-event statistics; applied in Phase 2).
- NA propagation from upstream pipeline steps.

NA propagation semantics: When `is_fitted = FALSE`, `model$abnormal_returns()` returns a tibble with `abnormal_returns = NA_real_` for all rows. All downstream test statistics that depend on fitted models then receive NA inputs and propagate NA to their outputs. The event is retained in the output tibble – it is never silently dropped.

Multi-event exclusion (2026-09-24 re-evaluation, item A2): A single-event's NA propagation above is not enough for cumulative (CAR-based) multi-event statistics (CSectT, PatellZ, Sign, GeneralizedSign, BMP, KolariPynnonen): an event with ZERO finite event-window abnormal returns must be EXCLUDED from every such statistic's group, not merely NA-propagated, because a cumulative sum that `coalesce()`s a fully-degenerate event's abnormal returns to 0 would silently understate the true cross-sectional dispersion. The exclusion is applied once per group (see `.exclude_all_na_events`) before any statistic is computed, and reported exactly once: silently if the excluded event's model already emitted the one contract warning guaranteed by CONTRACT-04 (an unfitted `ModelBase` instance), or via a single additional warning naming the excluded `event_id(s)` otherwise. Strict mode errors instead. This exclusion rule leaves the pre-existing STATS-03 convention for a PARTIAL gap unchanged: an event with at least one finite event-window abnormal return, but a missing value on some individual day, still contributes 0 to that event's CAR on the missing day(s) via `coalesce(abnormal_returns, 0)`.

Details

Degenerate-Input Contract for EventStudy Models

See Also

[ParameterSet](#), [MarketModel](#)

dieselgate

Volkswagen "Dieselgate" Multi-Automaker Event Study Dataset

Description

A small, frozen dataset bundling daily prices for four German automakers and the DAX benchmark index around the 2015 "dieselgate" emissions scandal, ready to drive a complete multi-group event study (`prepare_event_study()` -> `fit_model()` -> `calculate_statistics()`).

Usage

```
data(dieselgate)
```

Format

A named list with four elements:

firm A tibble of daily prices for all four automakers combined, with columns `symbol` (one of "VOW.DE", "PAH3.DE", "BMW.DE", "MBG.DE"), `date` (character, "%d.%m.%Y" format), and `adjusted` (numeric adjusted close). Rows for all firms are stacked (1 440 rows total).

index A tibble of DAX ("^GDAXI") daily prices with the same `symbol` / `date` / `adjusted` columns, used as the benchmark / reference market for all four events.

request A four-row tibble giving the event-study request specifications, one row per firm, with the nine columns expected by [EventStudyTask](#): `event_id` (1L to 4L), `firm_symbol`, `index_symbol`, `event_date` ("18.09.2015"), `group` ("VW Group" for VOW.DE/PAH3.DE, "Other" for BMW.DE/MBG.DE), `event_window_start` (-10), `event_window_end` (10), `shift_estimation_window` (-11), and `estimation_window_length` (250). `event_id` = 1 is VOW.DE for backward compatibility.

meta A list of provenance metadata: `firm_tickers` (character vector of all four tickers), `groups` (named list mapping group labels to tickers), `index_ticker`, `event_date`, `from`, `to`, `source`, `access_date`, and `note`.

Details

The event is the U.S. Environmental Protection Agency's Notice of Violation issued to Volkswagen on **2015-09-18** (a Friday); the share-price crash lands on the following trading days. The dataset covers two groups: the "VW Group" (VOW.DE, PAH3.DE – directly implicated firms) and "Other" (BMW.DE, MBG.DE – peer automakers). The bundled window layout uses a 250-trading-day estimation window ending 11 days before the event and an event window of $[-10, +10]$ trading days.

Fitting a market model on the "VW Group" events produces a strongly negative cumulative average abnormal return (CAAR approximately -39 roughly -17% and -13% average abnormal returns on the first two trading days after the disclosure, while the "Other" peer automakers show near-zero CAAR (approximately +1%), illustrating the idiosyncratic nature of the shock.

Firms:

- VOW.DE – Volkswagen AG ordinary shares (Xetra), event_id = 1
- PAH3.DE – Porsche Automobil Holding SE (Xetra), event_id = 2
- BMW.DE – BMW AG (Xetra), event_id = 3
- MBG.DE – Mercedes-Benz Group AG (Xetra), event_id = 4

Groups: "VW Group" (VOW.DE, PAH3.DE) vs "Other" (BMW.DE, MBG.DE). **Benchmark:** DAX performance index (ticker ^GDAXI). **Date range:** 2014-06-01 to 2015-11-01.

Source

Yahoo Finance daily adjusted prices, retrieved 2026-09-04 via the package's own [download_stock_data](#). This is a small illustrative sample bundled for academic / demonstration use only; see `data-raw/dieselgate.R` for the reproducible fetch script.

Examples

```
data(dieselgate)

# Build a multi-group task and run the full pipeline
task <- EventStudyTask$new(dieselgate$firm, dieselgate$index,
                           dieselgate$request)
task <- run_event_study(task, ParameterSet$new())

# Single-firm: VW crash (event_id = 1)
task$get_car(1L)

# Multi-group: CAAR comparison
vw_caar <- task$aar_caar_tbl[task$aar_caar_tbl$group == "VW Group", ]$SectT[[1]]
other_caar <- task$aar_caar_tbl[task$aar_caar_tbl$group == "Other", ]$SectT[[1]]
tail(vw_caar[, c("relative_index", "caar", "caar_t")], 1)
tail(other_caar[, c("relative_index", "caar", "caar_t")], 1)
```

download_factor_data *Download Factor Data*

Description

Downloads Fama-French factor data from the Kenneth French Data Library. No external package dependencies required (uses base R download + CSV parsing).

Usage

```
download_factor_data(
  model = c("ff3", "ff5", "mom"),
  frequency = c("daily", "monthly"),
  format_for_task = TRUE
)
```

Arguments

model	Factor model: "ff3" (default), "ff5", "mom" (momentum), or "ff3_monthly", "ff5_monthly".
frequency	Data frequency: "daily" (default) or "monthly".
format_for_task	Logical. If TRUE, formats for EventStudyTask factor_tbl.

Value

A tibble with factor data.

download_risk_free_rate

Download Risk-Free Rate

Description

Downloads risk-free rate data from the Kenneth French Data Library (extracted from the Fama-French factor files).

Usage

```
download_risk_free_rate(
  frequency = c("daily", "monthly"),
  source = "french",
  format_for_task = TRUE
)
```

Arguments

frequency	Data frequency: "daily" (default) or "monthly".
source	Data source: "french" (default).
format_for_task	Logical. If TRUE, formats for EventStudyTask.

Value

A tibble with date and risk_free_rate columns.

download_stock_data	<i>Download Stock Data</i>
---------------------	----------------------------

Description

Downloads historical stock price data for specified symbols. Uses **tidyquant** if available, otherwise falls back to **quantmod**.

Usage

```
download_stock_data(
  symbols,
  from,
  to = Sys.Date(),
  source = "yahoo",
  format_for_task = TRUE
)
```

Arguments

symbols	Character vector of stock ticker symbols.
from	Start date (Date or character in YYYY-MM-DD format).
to	End date. Default Sys.Date().
source	Data source: "yahoo" (default).
format_for_task	Logical. If TRUE, formats output for EventStudyTask: columns symbol, date (dd.mm.yyyy), adjusted.

Value

A tibble with stock data.

earnings_surprises	<i>Earnings Surprise Multi-Firm Event Study Dataset</i>
--------------------	---

Description

A small, frozen dataset bundling daily prices for three U.S. large-cap firms and the S&P 500 benchmark around the first-calendar-quarter 2023 (Jan-Mar) earnings announcements, reported late April / early May 2023, ready to drive a complete event study pipeline (prepare_event_study() -> fit_model() -> calculate_statistics()).

Usage

```
data(earnings_surprises)
```

Format

A named list with four elements:

firm A tibble of daily prices for all three firms combined, with columns `symbol` (one of "AAPL", "MSFT", "GOOGL"), `date` (character, "%d.%m.%Y" format), and `adjusted` (numeric adjusted close). Rows for all firms are stacked (813 rows total).

index A tibble of S&P 500 ("^GSPC") daily prices with the same `symbol` / `date` / `adjusted` columns, used as the benchmark for all three events.

request A three-row tibble giving the event-study request specifications, one row per firm, with the nine columns expected by [EventStudyTask](#): `event_id` (1L to 3L), `firm_symbol`, `index_symbol`, `event_date` ("04.05.2023" for AAPL, "25.04.2023" for MSFT and GOOGL), `group` ("Earnings Beat" for all firms), `event_window_start` (-5), `event_window_end` (5), `shift_estimation_window` (-6), and `estimation_window_length` (200).

meta A list of provenance metadata: `firm_tickers`, `index_ticker`, `event_dates`, `from`, `to`, `source`, `access_date`, and `note`.

Details

All three firms beat consensus EPS estimates for their respective January-to-March 2023 quarters: Apple Inc. reported on **2023-05-04** (its fiscal Q2 FY2023, which runs Jan-Mar) and beat by approximately 8%, driving a strong next-day return; Microsoft Corporation reported on **2023-04-25** (its fiscal Q3 FY2023, which runs Jan-Mar), beating cloud (Azure) estimates with a roughly +7% next-day response; Alphabet Inc. reported on **2023-04-25** (Q1 CY2023, Jan-Mar), with advertising revenue exceeding expectations. The bundled window layout uses a 200-trading-day estimation window ending 6 days before each event and an event window of [-5, +5] trading days. Running a market model on this single-group panel produces a positive cumulative average abnormal return (CAAR) over the event window, reflecting the shared earnings-beat signal across all three firms.

Firms:

- AAPL – Apple Inc. (NASDAQ), `event_id` = 1
- MSFT – Microsoft Corporation (NASDAQ), `event_id` = 2
- GOOGL – Alphabet Inc. Class A (NASDAQ), `event_id` = 3

Group: "Earnings Beat" (all firms). **Benchmark:** S&P 500 index (ticker ^GSPC). **Date range:** 2022-06-01 to 2023-06-30.

Source

Yahoo Finance daily adjusted prices, retrieved 2026-09-05 via the package's own [download_stock_data](#).

This is a small illustrative sample bundled for academic / demonstration use only; see `data-raw/earnings_surprises.R` for the reproducible fetch script.

Examples

```
data(earnings_surprises)

# Build task and run the full pipeline
task <- EventStudyTask$new(earnings_surprises$firm,
```

```

        earnings_surprises$index,
        earnings_surprises$request)
task <- run_event_study(task, ParameterSet$new())

# Multi-event: AAR/CAAR across all firms
caar_tbl <- task$aar_caar_tbl$CSectT[[1]]
tail(caar_tbl[, c("relative_index", "caar", "caar_t")], 1)

```

estimate_panel_event_study

Estimate Panel Event Study

Description

Estimate a panel event study using two-way fixed effects (TWFE) or dynamic TWFE. Returns event-time coefficient estimates suitable for event study plots.

Usage

```

estimate_panel_event_study(
  task,
  method = c("static_twfe", "dynamic_twfe", "sun_abraham", "callaway_santanna",
    "dechaisemartin_dhaultfoeuille", "borusyak_jaravel_spiess"),
  leads = 5,
  lags = 5,
  base_period = -1,
  cluster = NULL,
  verbose = getOption("eventstudy.verbose", TRUE),
  ...
)

```

Arguments

task	A PanelEventStudyTask.
method	Estimation method. One of "static_twfe", "dynamic_twfe", or "sun_abraham".
leads	Number of pre-treatment periods to include. Default 5.
lags	Number of post-treatment periods to include. Default 5.
base_period	The reference period (relative to treatment) to omit. Default -1 (the period just before treatment).
cluster	Name of the clustering variable for standard errors. Defaults to the unit ID.
verbose	Logical; if FALSE, suppress informational messages. Default getOption("eventstudy.verbose", TRUE).
...	Additional arguments passed to the underlying estimator (used by callaway_santanna, dechaisemartin_dhaultfoeuille, and borusyak_jaravel_spiess methods).

Value

The task with results populated, containing:

coefficients Tibble of event-time coefficients with std errors

model The fitted model object

method The estimation method used

See Also

Other eventstudy-tasks: [IntradayEventStudyTask](#), [PanelEventStudyTask](#), [SyntheticControlTask](#), [estimate_synthetic_control\(\)](#), [nonparametric_intraday_test\(\)](#), [plot_panel_event_study\(\)](#), [plot_synthetic_control\(\)](#), [prepare_intraday_event_study\(\)](#), [sc_placebo_test\(\)](#)

estimate_synthetic_control

Estimate Synthetic Control

Description

Estimate synthetic control weights by minimizing the pre-treatment mean squared prediction error. Weights are non-negative and sum to one.

Usage

```
estimate_synthetic_control(
  task,
  method = c("quadprog", "optim"),
  covariates = NULL,
  verbose = getOption("eventstudy.verbose", TRUE)
)
```

Arguments

task	A SyntheticControlTask.
method	Optimization method: "quadprog" (default, requires quadprog) or "optim" (uses stats::optim L-BFGS-B).
covariates	Optional character vector of covariate column names in both treated and donor data to include in the matching.
verbose	Logical; if FALSE, suppress informational messages. Default getOption("eventstudy.verbose", TRUE).

Value

The task with results populated.

See Also

Other eventstudy-tasks: [IntradayEventStudyTask](#), [PanelEventStudyTask](#), [SyntheticControlTask](#), [estimate_panel_event_study\(\)](#), [nonparametric_intraday_test\(\)](#), [plot_panel_event_study\(\)](#), [plot_synthetic_control\(\)](#), [prepare_intraday_event_study\(\)](#), [sc_placebo_test\(\)](#)

es_advise	<i>Grounded AI Advice for Event Study Results</i>
-----------	---

Description

Produces a grounded Advice S3 object by routing through a task-type dispatch, calling an optional LLM provider, parsing the JSON response, and running the runtime grounding guard – which drops any recommendation whose evidence[] cites a diagnostic key absent from the computed diagnostics or a value mismatching beyond numeric tolerance.

Usage

```
es_advise(  
  diagnostics,  
  task_type,  
  provider = NULL,  
  model = NULL,  
  section_hint = NULL,  
  ...  
)
```

Arguments

diagnostics	An es_diagnostics object returned by es_diagnostics() .
task_type	Character. One of "interpret", "recommend_stat", "recommend_model", "flag_robustness", "design_discussion", "report_writing".
provider	An optional provider R6 object (from provider()). Required for LLM-only task types. When NULL and task type is KB-based, falls back to the Phase 5 offline path.
model	Optional character model identifier. Reserved for forward compatibility: the effective model is the one the provider was constructed with (see provider()), so set the model there. Accepted here without error so calling code can pass it, but it does not override the provider’s configured model.
section_hint	Optional character scalar naming the report section to scope the narrative to (e.g. "exec_summary", "results", "robustness"). NULL (default) leaves the prompt byte-identical to its pre-Phase-18 form. Only applied for the "report_writing" task type.
...	Additional arguments (currently ignored; reserved for future use).

Details

Grounding guarantee: Every returned recommendation is provably tied to a value the package actually computed (`es_diagnostics()`). The guard is enforced in R, independent of the prompt (ADV-04).

Task type routing:

`recommend_stat`, `flag_robustness` No provider: returns the Phase 5 `es_advise` object (offline KB path – deterministic, `is_deterministic` = TRUE). With provider: KB produces grounded evidence[], LLM adds prose; returns an Advice object (`is_deterministic` = FALSE).

`interpret`, `recommend_model`, `design_discussion`, `report_writing` LLM-required: `stop()` when `provider` = NULL (ADV-06). With provider: LLM produces full advice; guard runs.

Failure discipline: Any provider failure, malformed JSON, or empty response degrades to one `warning()` + an empty Advice object – never a crash, never a fabricated result (mirrors `.handle_degenerate()`).

Value

For KB task types without a provider: an `es_advise` S3 object (Phase 5 offline path, `is_deterministic` = TRUE). For all other paths: an Advice S3 object with fields `source`, `is_deterministic`, `task_type`, `interpretation`, `recommendations`, `caveats`, `n_dropped`.

See Also

[es_diagnostics](#), [recommend_stat](#), [flag_robustness](#), [provider](#), [run_event_study](#)

Other eventstudy-advisor: [AnthropicProvider](#), [CustomProvider](#), [OpenAICompatProvider](#), [ProviderBase](#), [es_diagnostics\(\)](#), [es_kb\(\)](#), [flag_robustness\(\)](#), [provider\(\)](#), [recommend_stat\(\)](#)

Examples

Offline KB path (no LLM, no network, no credentials):

```
data(dieselgate)
task <- EventStudyTask$new(dieselgate$firm, dieselgate$index, dieselgate$request)
task <- run_event_study(task, ParameterSet$new())
diag <- es_diagnostics(task)
advice_kb <- es_advise(diag, task_type = "recommend_stat")
print(advice_kb) # es_advise S3
```

LLM-grounded path (requires a network connection and provider credentials):

Not run:

```
p <- provider("openai")
advice <- es_advise(diag, task_type = "recommend_stat", provider = p)
print(advice) # Advice S3 with grounding guarantee
```

End(Not run)

es_colours

EventStudy Colour Palette

Description

A named character vector of hex colours for EventStudy plots, anchored on the Okabe-Ito colorblind-safe (Color Universal Design) qualitative set with the EventStudy brand primary blue as the leading colour.

Usage

```
es_colours
```

Format

A named character vector of length 12. Every value is a six-digit hex colour (`^#[0-9A-Fa-f]{6}$`).

Details

Semantic roles: `primary` (main series line/point/fill), `event` (event-date marker), `reference` (zero lines, ACF baseline), `ci_band` (confidence ribbon fill; apply alpha at the geom level), and `group1-group8` (qualitative multi-series palette in Okabe-Ito order).

See Also

[theme_eventstudy](#), [plot_event_study](#)

Other eventstudy-plots: [plot_car_distribution\(\)](#), [plot_diagnostics\(\)](#), [plot_event_study\(\)](#), [plot_stocks\(\)](#), [theme_eventstudy\(\)](#)

es_diagnostics

Harvest Diagnostics from a Fitted EventStudyTask

Description

Extracts already-computed statistical signals from a fitted EventStudyTask into a flat, JSON-ready S3 list. This is the always-available grounding foundation for the knowledge base and offline advice layer – it recomputes nothing; all signals already exist in the fitted task.

Usage

```
es_diagnostics(task, max_events = 20L)
```

Arguments

task	A fitted EventStudyTask (after <code>fit_model()</code> and optionally <code>calculate_statistics()</code> have been called).
max_events	Integer. Maximum number of events to include in the per-event sections (<code>estimation_window</code> , <code>event_window</code> , <code>contract_state</code>). Events are ranked by anomaly score (degenerate events first, then by absolute final CAR). The remainder is summarised in <code>aggregate_summary</code> . Default: 20.

Value

A named list of class "es_diagnostics" with six sections:

meta	List: <code>n_events_total</code> (integer), <code>n_events_shown</code> (integer), <code>n_events_summarized</code> (integer), <code>event_ids_shown</code> (integer vector).
estimation_window	List of plain numeric vectors (length <code>n_events_shown</code>): <code>r2</code> , <code>sigma</code> , <code>degree_of_freedom</code> , <code>shapiro_p</code> , <code>dw_stat</code> , <code>ljung_box_p</code> , <code>acf1</code> . Each entry is <code>NA_real_</code> when the model was not fitted or residuals are insufficient for the test.
event_window	List of plain numeric vectors (length <code>n_events_shown</code>): <code>ar_t</code> (last event-window day AR t-stat), <code>ar_p</code> (two-sided p-value), <code>car_t</code> (full-window CAR t-stat), <code>car_p</code> (two-sided p-value), <code>final_car</code> . All <code>NA_real_</code> when the ART/CART columns are absent.
cross_sectional	List of scalars aggregated across ALL events: <code>n_events</code> (total), <code>n_valid_events</code> (fitted count), <code>car_iqr</code> , <code>car_sd</code> , <code>n_overlap_pairs</code> , <code>any_overlap</code> . Multi-event fields degrade to <code>NA</code> when <code>task\$aar_caar_tbl</code> is <code>NULL</code> .
contract_state	List of logical/numeric vectors (length <code>n_events_shown</code>): <code>is_fitted</code> , <code>na_ar_count</code> , <code>na_est_count</code> , <code>insufficient_obs</code> , <code>zero_var_index</code> .
aggregate_summary	Named list summarising remainder events (those beyond <code>max_events</code>): <code>n_summarized</code> , <code>mean_r2</code> , <code>median_r2</code> , <code>mean_final_car</code> , <code>n_fitted</code> , <code>n_degenerate</code> . <code>NULL</code> when no events are summarised (all shown).

See Also

[model_diagnostics](#), [recommend_stat](#), [flag_robustness](#), [run_event_study](#)

Other eventstudy-advisor: [AnthropicProvider](#), [CustomProvider](#), [OpenAICompatProvider](#), [ProviderBase](#), [es_advise\(\)](#), [es_kb\(\)](#), [flag_robustness\(\)](#), [provider\(\)](#), [recommend_stat\(\)](#)

Description

Returns `EVENTSTUDY_KB`, the package-level pure-R list of rule records that maps diagnostic conditions to grounded methodological recommendations. Each rule carries a structured academic citation and can be evaluated against an [es_diagnostics](#) object.

Usage

```
es_kb()
```

Details

The KB is the correctness-critical layer consumed by [recommend_stat](#) and [flag_robustness](#), and exported here so that Phase 7 can inject its contents into an LLM system prompt without accessing an internal package object.

KB-04 note: This function delivers the *structure* only – exported and serializable, ready for Phase 7 system-prompt injection. The actual prompt-injection behavior is a Phase 7 deliverable and is deliberately out of scope here.

Value

A named list of rule records, each with fields: id, category, condition, recommendation, citation (list of author/year/key/venue), and severity.

Note

Each rule’s condition field is an R function closure and is therefore **not** JSON-serializable. Any consumer that serializes the KB (e.g. Phase 7 system-prompt injection) must drop the condition field from each rule record before encoding to JSON.

See Also

[recommend_stat](#), [flag_robustness](#), [es_diagnostics](#)

Other eventstudy-advisor: [AnthropicProvider](#), [CustomProvider](#), [OpenAICompatProvider](#), [ProviderBase](#), [es_advise\(\)](#), [es_diagnostics\(\)](#), [flag_robustness\(\)](#), [provider\(\)](#), [recommend_stat\(\)](#)

Examples

```
kb <- es_kb()
length(kb)           # number of rules
kb[[1]]$id           # id of first rule
kb[[1]]$citation      # citation list
```

es_report

One-Call Event Study Report Orchestrator

Description

Generates a complete event study report in a single call: deep-clones the task (non-mutation guarantee, REPORT-04), harvests diagnostics via [es_diagnostics\(\)](#), and delegates rendering to [generate_report\(\)](#). Returns the output file path(s) VISIBLY so the path prints at the REPL without an explicit `print()`.

Usage

```

es_report(
  task,
  output_file = "event_study_report.html",
  format = "html",
  sections = c("exec_summary", "data_methods", "results", "diagnostics", "robustness",
    "references"),
  provider = NULL,
  title = "Event Study Report",
  author = NULL,
  confidence_level = 0.95,
  interactive = TRUE,
  verbose = getOption("eventstudy.verbose", TRUE),
  ...
)

```

Arguments

<code>task</code>	A fitted <code>EventStudyTask</code> or <code>PanelEventStudyTask</code> .
<code>output_file</code>	Output file path (extension overridden per format). Default <code>"event_study_report.html"</code> .
<code>format</code>	Character vector of output formats, any subset of <code>c("html", "pdf", "word", "md")</code> . Default <code>"html"</code> . Multiple formats render one file per format sharing a common basename.
<code>sections</code>	Character vector of sections to include. Default is all six fixed sections: <code>c("exec_summary", "data_metho</code>
<code>provider</code>	Optional LLM provider forwarded to <code>generate_report()</code> . When <code>NULL</code> (default), renders a fully offline report (no API key, no network required).
<code>title</code>	Report title string.
<code>author</code>	Author name (optional, default <code>NULL -> ""</code>).
<code>confidence_level</code>	Confidence level for plots. Default 0.95.
<code>interactive</code>	Logical. Use interactive plotly plots in HTML output. Default <code>TRUE</code> .
<code>verbose</code>	Logical; if <code>FALSE</code> , suppress informational messages. Default <code>getOption("eventstudy.verbose", TRUE)</code> .
<code>...</code>	Additional arguments forwarded to <code>generate_report()</code> and on to <code>rmarkdown::render</code> .

Details

The narrative is assembled ONCE inside `generate_report()` (NARR-01): `es_report()` does NOT call `assemble_report_narrative()` or `es_advise()` directly. A provider argument is forwarded to `generate_report()`, which assembles the narrative once before the format loop. When `provider = NULL` (default), the report is fully offline with no network or API key required.

Value

A named character vector of output file path(s), one per rendered format, returned **visibly** (the path prints at the REPL). Keyed by format name, e.g. `c(html = "/tmp/report.html")`. A single-format call returns a length-1 named vector; `result[["html"]]` and `result[[1L]]` both resolve.

See Also

[generate_report](#), [es_diagnostics](#), [run_event_study](#)

Other eventstudy-export: [export_results\(\)](#), [generate_report\(\)](#), [tidy.EventStudyTask\(\)](#)

Examples

```
data(dieselgate)
task <- EventStudyTask$new(dieselgate$firm, dieselgate$index, dieselgate$request)
task <- run_event_study(task, ParameterSet$new())

# Offline single-format report written to a temporary directory
out <- file.path(tempdir(), "event_study_report.html")
path <- es_report(task, output_file = out, format = "html")
path[["html"]]
```

eventstudy-deprecation

EventStudy Deprecation Policy

Description

Policy: warn, never silently break.

Deprecated argument names and function names in EventStudy continue to work and emit exactly one deprecation warning per call, pointing the user to the replacement. Deprecated surfaces are never removed in a patch or minor release; removal (the "defunct" stage) is reserved for a future major version and will be announced in NEWS.md.

Mechanism: Every renamed public argument has an explicit old-name shim that calls `.deprecate_arg()` and then forwards the supplied value under the new name. The shim fires before any computation, so the old name is fully-equivalent to the new name for all valid inputs.

Warning source: Deprecation warnings always use base `.Deprecated()` so they work without any additional package dependency. When the optional **lifecycle** package is installed (Suggests), the warning is also routed through `lifecycle::deprecate_warn()` for richer IDE integration; **lifecycle** is never a hard Imports dependency.

NEWS discipline: Every deprecation and rename is documented in NEWS.md under the release heading in which it was introduced.

See Also

[run_event_study](#), [plot_stocks](#)

eventstudy-shape-contracts

Return-Shape Contracts for EventStudy Pipeline Results

Description

The shape-contract system locks the column names and types of the EventStudy pipeline's result tibbles against accidental structural drift. It is designed for development and CI use only.

Default: OFF. When the option is unset (the default) the entire shape-contract path is a strict no-op — every existing caller and the full test suite are completely unaffected.

Opt-in: `options(EventStudy.shape_contracts = TRUE)`

Philosophy (mirrors the degenerate-input contract): On a column-name or type mismatch the contract emits exactly *one* `warning()`, naming the context and the specific drift, and then returns invisibly. It **never** calls `stop()`. The correctly-shaped `is_fitted = FALSE` degenerate output (same column names, NA-propagated values) is treated as a *valid* shape and produces no warning.

Coverage:

- Single-event statistics tibbles: ART (ARTTest) and CART (CARTTest), including their `is_fitted = FALSE` degenerate variants.
- Multi-event AAR/CAAR statistics tibble: CSectT (CSectTTest) and all other multi-event result tibbles, including their `is_fitted = FALSE` degenerate variants.

Configuration:

- Via package option: `options(EventStudy.shape_contracts = TRUE)`
- Default is FALSE (off).

Details

Return-Shape Contracts for EventStudy Pipeline Results

See Also

[degenerate-input-contract](#), [ParameterSet](#)

EventStudyTask

Initialization of an Event Study task.

Description

Event Study Task

The Event Study task contains all necessary data for performing an event study. Furthermore, all calculations are saved in the internal dataframe named `data_tbl`.

Public fields

`data_tbl` All calculations are saved in this dataframe. The dataframe is a nested object with each row is one event, identified by the event id, the group, and the firm symbol.

`aar_caar_tbl` Placeholder for AAR and CAAR test statistics.

`.keys` event identifier. Do not change.

`.index` The time column name

`.target` The price column name.

`.request_file_columns` The necessary column names of the request dataframe. `estimation_window_length` here is an input count (number of trading days before the event window); it is distinct from the derived 0/1 `estimation_window` flag column that `prepare_event_study()` later adds to each event's nested data.

`factor_tbl` Optional factor data for multi-factor models (Fama-French, Carhart). Must contain a date column.

Active bindings

`symbols` Read-only field to get the firm symbols.

`symbol_data` Read-only field to get the firm symbol data.

`group_level_data` Read-only field to get the group data.

Methods**Public methods:**

- `EventStudyTask$new()`
- `EventStudyTask$print()`
- `EventStudyTask$summary()`
- `EventStudyTask$get_ar()`
- `EventStudyTask$get_car()`
- `EventStudyTask$get_aar()`
- `EventStudyTask$get_model_stats()`
- `EventStudyTask$clone()`

`EventStudyTask$new()`: Create a new `EventStudyTask` from stock data, reference market data, and an event request specification.

Usage:

```
EventStudyTask$new(
  firm_stock_data_tbl,
  reference_tbl,
  request_tbl,
  factor_tbl = NULL
)
```

Arguments:

`firm_stock_data_tbl` Dataframe with firm stock data. This dataframe must contain the stock (col: symbol) the date (col: date) and the price (col: adjusted) column.

reference_tbl Dataframe with firm reference data.
request_tbl The request dataframe for each event.
factor_tbl Optional dataframe with factor data (e.g., Fama-French factors). Must contain a date column plus factor columns.

EventStudyTask\$print(): Print a summary of the EventStudyTask.

Usage:

```
EventStudyTask$print(...)
```

EventStudyTask\$summary(): Summarize the event study results.

Usage:

```
EventStudyTask$summary()
```

Returns: A list with summary information about the event study results.

EventStudyTask\$get_ar(): Extract abnormal returns for a single event.

Usage:

```
EventStudyTask$get_ar(event_id = NULL)
```

Arguments:

event_id The event identifier.

Returns: A tibble with abnormal returns for the event window.

EventStudyTask\$get_car(): Extract cumulative abnormal returns for a single event.

Usage:

```
EventStudyTask$get_car(event_id = NULL)
```

Arguments:

event_id The event identifier.

Returns: A tibble with cumulative abnormal returns for the event window.

EventStudyTask\$get_aar(): Extract average abnormal returns across events.

Usage:

```
EventStudyTask$get_aar(group = NULL, stat_name = "CSectT")
```

Arguments:

group Optional group name to filter by.

stat_name Name of the multi-event test statistic to extract. Defaults to "CSectT".

Returns: A tibble with AAR/CAAR results.

EventStudyTask\$get_model_stats(): Extract model statistics for a single event.

Usage:

```
EventStudyTask$get_model_stats(event_id = NULL)
```

Arguments:

event_id The event identifier.

Returns: A list with model statistics.

EventStudyTask\$clone(): The objects of this class are cloneable with this method.

Usage:

EventStudyTask\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

See Also

Other eventstudy-pipeline: [ParameterSet](#), [calculate_statistics\(\)](#), [fit_model\(\)](#), [prepare_event_study\(\)](#), [run_event_study\(\)](#)

export_results	<i>Export Event Study Results</i>
----------------	-----------------------------------

Description

Export event study results to CSV, Excel, or LaTeX formats.

Usage

```
export_results(  
  task,  
  file,  
  format = NULL,  
  which = c("ar", "car", "aar", "model"),  
  stat_name = "CSectT",  
  ...  
)
```

Arguments

task	A fitted EventStudyTask with statistics computed.
file	Path to the output file. The file extension determines the format unless format is explicitly specified.
format	Output format: "csv", "xlsx", or "latex". If NULL, inferred from the file extension.
which	Which results to export. One or more of "ar", "car", "aar", "model". Defaults to all available.
stat_name	Name of the multi-event test statistic to use for AAR/CAAR tables. Defaults to "CSectT".
...	Additional arguments passed to the format-specific writer.

Value

The file path (invisibly).

See Also

[run_event_study](#), [tidy.EventStudyTask](#)

Other eventstudy-export: [es_report\(\)](#), [generate_report\(\)](#), [tidy.EventStudyTask\(\)](#)

FamaFrench3FactorModel

Fama-French Three-Factor Model

Description

Implements the Fama and French (1993) three-factor model:

$$R_i - R_f = \alpha + \beta_m(R_m - R_f) + \beta_s SMB + \beta_h HML + \epsilon$$

The data must contain columns: `excess_return` (firm return minus risk-free), `market_excess` (market return minus risk-free), `smb`, and `hml`. These can be joined via a factor table in `EventStudyTask`.

Super classes

[ModelBase](#) -> [LinearFactorModel](#) -> `FamaFrench3FactorModel`

Public fields

`model_name` Name of the model.

`formula` The three-factor regression formula.

`required_columns` Required data columns.

Methods**Public methods:**

- [FamaFrench3FactorModel\\$new\(\)](#)
- [FamaFrench3FactorModel\\$abnormal_returns\(\)](#)
- [FamaFrench3FactorModel\\$clone\(\)](#)

`FamaFrench3FactorModel$new()`: Create a new `FamaFrench3FactorModel`.

Usage:

```
FamaFrench3FactorModel$new(use_hac = FALSE, hac_lag = NULL)
```

Arguments:

`use_hac` Logical. Use HAC (Newey-West) standard errors.

`hac_lag` Integer or NULL. Lag truncation for Newey-West.

`FamaFrench3FactorModel$abnormal_returns()`: Calculate abnormal returns using the three-factor model.

Usage:

```
FamaFrench3FactorModel$abnormal_returns(data_tbl)
```

Arguments:

data_tbl Data frame or tibble.

FamaFrench3FactorModel\$clone(): The objects of this class are cloneable with this method.

Usage:

```
FamaFrench3FactorModel$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other eventstudy-models: [BHARModel](#), [Carhart4FactorModel](#), [ComparisonPeriodMeanAdjustedModel](#), [DCCGARCHModel](#), [FamaFrench5FactorModel](#), [GARCHModel](#), [LinearFactorModel](#), [LogReturn](#), [MarketAdjustedModel](#), [MarketModel](#), [ModelBase](#), [ReturnCalculation](#), [RollingWindowModel](#), [SimpleReturn](#), [VolatilityModel](#), [VolumeModel](#)

FamaFrench5FactorModel

Fama-French Five-Factor Model

Description

Implements the Fama and French (2015) five-factor model:

$$R_i - R_f = \alpha + \beta_m(R_m - R_f) + \beta_s SMB + \beta_h HML + \beta_r RMW + \beta_c CMA + \epsilon$$

Requires columns: excess_return, market_excess, smb, hml, rmw, cma.

Super classes

[ModelBase](#) -> [LinearFactorModel](#) -> FamaFrench5FactorModel

Public fields

model_name Name of the model.

formula The five-factor regression formula.

required_columns Required data columns.

Methods

Public methods:

- [FamaFrench5FactorModel\\$new\(\)](#)
- [FamaFrench5FactorModel\\$abnormal_returns\(\)](#)
- [FamaFrench5FactorModel\\$clone\(\)](#)

`FamaFrench5FactorModel$new()`: Create a new `FamaFrench5FactorModel`.

Usage:

`FamaFrench5FactorModel$new(use_hac = FALSE, hac_lag = NULL)`

Arguments:

`use_hac` Logical. Use HAC (Newey-West) standard errors.
`hac_lag` Integer or NULL. Lag truncation for Newey-West.

`FamaFrench5FactorModel$abnormal_returns()`: Calculate abnormal returns using the five-factor model.

Usage:

`FamaFrench5FactorModel$abnormal_returns(data_tbl)`

Arguments:

`data_tbl` Data frame or tibble.

`FamaFrench5FactorModel$clone()`: The objects of this class are cloneable with this method.

Usage:

`FamaFrench5FactorModel$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-models: [BHARModel](#), [Carhart4FactorModel](#), [ComparisonPeriodMeanAdjustedModel](#), [DCCGARCHModel](#), [FamaFrench3FactorModel](#), [GARCHModel](#), [LinearFactorModel](#), [LogReturn](#), [MarketAdjustedModel](#), [MarketModel](#), [ModelBase](#), [ReturnCalculation](#), [RollingWindowModel](#), [SimpleReturn](#), [VolatilityModel](#), [VolumeModel](#)

fit_model	<i>Train the defined model on each event and calculate the abnormal return.</i>
-----------	---

Description

Train the defined model on each event and calculate the abnormal return.

Usage

`fit_model(task, parameter_set)`

Arguments

task	The event study task
parameter_set	The parameter set that defines the event study.

Value

task

See Also

[run_event_study](#), [prepare_event_study](#), [calculate_statistics](#)

Other eventstudy-pipeline: [EventStudyTask](#), [ParameterSet](#), [calculate_statistics\(\)](#), [prepare_event_study\(\)](#), [run_event_study\(\)](#)

flag_robustness	<i>Flag Robustness Issues via Offline KB Matching</i>
-----------------	---

Description

Evaluates the KB decision table (category "robustness" rules) against the diagnostic signals extracted from a fitted EventStudyTask or a precomputed es_diagnostics object, and returns a severity-ranked es_advice S3 object.

Usage

```
flag_robustness(x, provider = NULL, ...)
```

```
## Default S3 method:
```

```
flag_robustness(x, provider = NULL, ...)
```

```
## S3 method for class 'EventStudyTask'
```

```
flag_robustness(x, provider = NULL, ...)
```

```
## S3 method for class 'es_diagnostics'
```

```
flag_robustness(x, provider = NULL, ...)
```

Arguments

x	A fitted EventStudyTask (after fit_model()) or a precomputed es_diagnostics object returned by es_diagnostics().
provider	Accepted but ignored in the offline path – present only so the Phase 7 call shape is forward-compatible. Default NULL.
...	Additional arguments (currently ignored).

Details

No LLM provider, network connection, or API key is required. The returned `es_advice` object has the same shape as the Phase 7 Advice contract, flagged `is_deterministic = TRUE` and `source = "offline_kb"`.

Value

An S3 object of class `"es_advice"` – see [recommend_stat](#) for field descriptions. Rules are filtered to `category == "robustness"`.

See Also

[recommend_stat](#), [es_diagnostics](#), [es_kb](#)

Other eventstudy-advisor: [AnthropicProvider](#), [CustomProvider](#), [OpenAICompatProvider](#), [ProviderBase](#), [es_advise\(\)](#), [es_diagnostics\(\)](#), [es_kb\(\)](#), [provider\(\)](#), [recommend_stat\(\)](#)

Examples

```
data(dieselgate)
task <- EventStudyTask$new(dieselgate$firm, dieselgate$index, dieselgate$request)
task <- run_event_study(task, ParameterSet$new())
advice <- flag_robustness(task)
print(advice)
```

format.Advice

Format method for Advice objects

Description

Builds the character vector rendered by `print.Advice` (one element per output line). The opt-in Advisor Pro footer is captured here (a no-op unless `options(eventstudy.advisor_pro_footer = TRUE)`), so `print.Advice` must not re-emit it.

Usage

```
## S3 method for class 'Advice'
format(x, ...)
```

Arguments

`x` An object of class `"Advice"`.
`...` Additional arguments (ignored).

Value

A character vector, one element per printed line.

format.es_advice	<i>Format method for es_advice objects</i>
------------------	--

Description

Builds the character vector rendered by `print.es_advice` (one element per output line). The opt-in Advisor Pro footer is captured here (a no-op unless `options(eventstudy.advisor_pro_footer = TRUE)`), so `print.es_advice` must not re-emit it.

Usage

```
## S3 method for class 'es_advice'
format(x, ...)
```

Arguments

x	An object of class "es_advice".
...	Additional arguments (ignored).

Value

A character vector, one element per printed line.

format.es_cross_sectional	<i>Format Cross-Sectional Regression Results</i>
---------------------------	--

Description

Builds the character vector rendered by `print.es_cross_sectional` (one element per output line). The coefficient table is rendered via `base::print.data.frame` and spliced in at the same position.

Usage

```
## S3 method for class 'es_cross_sectional'
format(x, ...)
```

Arguments

x	An es_cross_sectional object.
...	Additional arguments (unused).

Value

A character vector, one element per printed line.

format.es_diagnostics *Format method for es_diagnostics objects*

Description

Builds the character vector rendered by `print.es_diagnostics` (one element per output line). See [print.es_diagnostics](#).

Usage

```
## S3 method for class 'es_diagnostics'  
format(x, ...)
```

Arguments

x An object of class "es_diagnostics".
... Additional arguments (ignored).

Value

A character vector, one element per printed line.

format.es_simulation *Format method for es_simulation objects*

Description

Builds the character vector rendered by `print.es_simulation` (one element per output line).

Usage

```
## S3 method for class 'es_simulation'  
format(x, ...)
```

Arguments

x An object of class "es_simulation".
... Additional arguments (unused).

Value

A character vector, one element per printed line.

```
format.EventStudySummary
```

Format method for EventStudySummary

Description

Builds the character vector rendered by `print.EventStudySummary` (one element per output line). See [print.EventStudySummary](#).

Usage

```
## S3 method for class 'EventStudySummary'
format(x, ...)
```

Arguments

<code>x</code>	An EventStudySummary object.
<code>...</code>	Additional arguments (unused).

Value

A character vector, one element per printed line.

GARCHModel

GARCH Model

Description

Event study model using GARCH(1,1) for time-varying volatility estimation. Uses the **rugarch** package to fit a GARCH(1,1) model with a market return regressor in the mean equation during the estimation window. Abnormal returns are computed as the difference between observed returns and the GARCH conditional mean. The time-varying sigma from GARCH can be used for standardized test statistics.

Super class

[ModelBase](#) -> GARCHModel

Public fields

<code>model_name</code>	Name of the model.
<code>garch_order</code>	GARCH order as <code>c(p, q)</code> . Default <code>c(1,1)</code> .

Methods

Public methods:

- [GARCHModel\\$fit\(\)](#)
- [GARCHModel\\$abnormal_returns\(\)](#)
- [GARCHModel\\$clone\(\)](#)

[GARCHModel\\$fit\(\)](#): Fit the GARCH model on the estimation window.

Usage:

```
GARCHModel$fit(data_tbl)
```

Arguments:

`data_tbl` Data frame or tibble with `firm_returns`, `index_returns`, `estimation_window`, `event_window` columns.

[GARCHModel\\$abnormal_returns\(\)](#): Calculate abnormal returns from the GARCH model.

Usage:

```
GARCHModel$abnormal_returns(data_tbl)
```

Arguments:

`data_tbl` Data frame or tibble.

[GARCHModel\\$clone\(\)](#): The objects of this class are cloneable with this method.

Usage:

```
GARCHModel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-models: [BHARModel](#), [Carhart4FactorModel](#), [ComparisonPeriodMeanAdjustedModel](#), [DCCGARCHModel](#), [FamaFrench3FactorModel](#), [FamaFrench5FactorModel](#), [LinearFactorModel](#), [LogReturn](#), [MarketAdjustedModel](#), [MarketModel](#), [ModelBase](#), [ReturnCalculation](#), [RollingWindowModel](#), [SimpleReturn](#), [VolatilityModel](#), [VolumeModel](#)

GeneralizedSignTest	<i>Generalized Sign Test (Cowan 1992)</i>
---------------------	---

Description

Adjusts the sign test for the expected proportion of positive abnormal returns estimated from the estimation window, rather than assuming 0.5. This accounts for asymmetry in the return distribution.

Super class

[TestStatisticBase](#) -> GeneralizedSignTest

Public fields

name Short code of the test statistic.

Methods**Public methods:**

- [GeneralizedSignTest\\$compute\(\)](#)
- [GeneralizedSignTest\\$clone\(\)](#)

[GeneralizedSignTest\\$compute\(\)](#): Computes the generalized sign test for multiple events.

Usage:

```
GeneralizedSignTest$compute(data_tbl, model)
```

Arguments:

data_tbl The data for a multiple event with calculated abnormal returns.

model The fitted model (unused).

[GeneralizedSignTest\\$clone\(\)](#): The objects of this class are cloneable with this method.

Usage:

```
GeneralizedSignTest$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other eventstudy-statistics: [ARTTest](#), [BHARTTest](#), [BMPTTest](#), [CARTTest](#), [CSectTTest](#), [CalendarTimePortfolioTest](#), [KolariPynnonenTest](#), [MultiEventStatisticsSet](#), [PatellZTest](#), [RankTest](#), [SignTest](#), [SingleEventStatisticsSet](#), [StatisticsSetBase](#), [TestStatisticBase](#), [adjust_p_values\(\)](#), [bootstrap_test\(\)](#), [car_by_group\(\)](#), [car_quantiles\(\)](#), [cross_sectional_regression\(\)](#), [model_diagnostics\(\)](#), [pretrend_test\(\)](#), [simulate_event_study\(\)](#), [validate_task\(\)](#)

generate_report

Generate Event Study Report (Multi-Format)

Description

Renders an automated report from a completed event study task in one or more output formats. Uses a bundled RMarkdown template with configurable sections. The narrative is assembled ONCE before the format loop – `es_advise` is never called per format (NARR-01).

Usage

```

generate_report(
  task,
  output_file = "event_study_report.html",
  format = "html",
  title = "Event Study Report",
  author = NULL,
  sections = c("exec_summary", "data_methods", "results", "diagnostics", "robustness",
    "references"),
  cross_sectional = NULL,
  confidence_level = 0.95,
  interactive = TRUE,
  advice = NULL,
  narrative = NULL,
  provider = NULL,
  verbose = getOption("eventstudy.verbose", TRUE),
  ...
)

```

Arguments

task	A fitted EventStudyTask or PanelEventStudyTask.
output_file	Output file path (extension overridden per format). Default "event_study_report.html".
format	Character vector of output formats, any subset of c("html", "pdf", "word", "md"). Default "html". Multiple formats render one file per format sharing a common basename. Missing optional toolchains emit one message() each and are skipped; only inability to render HTML raises stop().
title	Report title string.
author	Author name (optional, default "").
sections	Character vector of sections to include. Default is all six fixed sections: c("exec_summary", "data_metho
cross_sectional	Optional cross-sectional regression results to include.
confidence_level	Confidence level for plots. Default 0.95.
interactive	Logical. Use interactive plotly plots in HTML output. Default TRUE.
advice	An optional grounded Advice object returned by es_advise (task_type = "report_writing"). When supplied and valid, renders an AI Advisor Interpretation section. A supplied but invalid advice is silently coerced to NULL with one warning() – the report is never broken.
narrative	An optional named list keyed by section (e.g. list(exec_summary = "...", data_methods = "...", results = "...", robustness = "...")), typically from assemble_report_narrative(). When NULL (default), the render output is byte-identical to the v0.63.x baseline. A supplied but invalid narrative is silently coerced to NULL with one warning().

provider	Optional LLM provider passed to <code>assemble_report_narrative()</code> when narrative is NULL. If both provider and narrative are NULL, a fully offline narrative is assembled. Ignored when a pre-built narrative list is supplied by the caller.
verbose	Logical; if FALSE, suppress informational messages. Default <code>getOption("eventstudy.verbose", TRUE)</code> .
...	Additional arguments passed to <code>rmarkdown::render</code> .

Value

A named character vector of output file paths, invisibly, keyed by format name (e.g. `c(html = "/tmp/report.html", pdf = "/tmp/report.pdf")`). A single-format call returns a length-1 named vector: `result[["html"]]` and `result[[1L]]` both resolve (backward-compatible with prior single-path callers).

See Also

Other eventstudy-export: [es_report\(\)](#), [export_results\(\)](#), [tidy.EventStudyTask\(\)](#)

Examples

```
data(dieselgate)
task <- EventStudyTask$new(dieselgate$firm, dieselgate$index, dieselgate$request)
task <- run_event_study(task, ParameterSet$new())

# Single HTML format written to a temporary directory
out <- file.path(tempdir(), "event_study_report.html")
path <- generate_report(task, output_file = out, format = "html")
path[["html"]]
```

IntradayEventStudyTask

Intraday Event Study Task

Description

Extension of [EventStudyTask](#) for intraday (high-frequency) event studies. Supports POSIXct timestamps and time-based windows specified in minutes or hours.

Differences from EventStudyTask

- Date column contains POSIXct timestamps instead of date strings
- Event windows specified as time offsets (e.g., `"-30min"`, `"+60min"`)
- Handles trading session boundaries

Super class

[EventStudyTask](#) -> IntradayEventStudyTask

Public fields

.index The time column name (POSIXct).
 .target The price column name.
 .request_file_columns Required columns for intraday requests.

Methods**Public methods:**

- [IntradayEventStudyTask\\$new\(\)](#)
- [IntradayEventStudyTask\\$print\(\)](#)
- [IntradayEventStudyTask\\$clone\(\)](#)

[IntradayEventStudyTask\\$new\(\)](#): Create a new [IntradayEventStudyTask](#).

Usage:

```
IntradayEventStudyTask$new(
  firm_stock_data_tbl,
  reference_tbl,
  request_tbl,
  factor_tbl = NULL
)
```

Arguments:

`firm_stock_data_tbl` Dataframe with firm intraday data. Must contain columns: symbol, timestamp (POSIXct), and price.
`reference_tbl` Dataframe with reference market intraday data.
`request_tbl` Request dataframe with event_timestamp (POSIXct), and window specifications in minutes.
`factor_tbl` Optional factor data.

[IntradayEventStudyTask\\$print\(\)](#): Print summary.

Usage:

```
IntradayEventStudyTask$print(...)
```

[IntradayEventStudyTask\\$clone\(\)](#): The objects of this class are cloneable with this method.

Usage:

```
IntradayEventStudyTask$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-tasks: [PanelEventStudyTask](#), [SyntheticControlTask](#), [estimate_panel_event_study\(\)](#), [estimate_synthetic_control\(\)](#), [nonparametric_intraday_test\(\)](#), [plot_panel_event_study\(\)](#), [plot_synthetic_control\(\)](#), [prepare_intraday_event_study\(\)](#), [sc_placebo_test\(\)](#)

KolariPynnonenTest *Kolari-Pynnoenen Adjusted BMP Test*

Description

Adjusts the BMP (Boehmer, Musumeci, Poulsen 1991) test for cross-sectional correlation of abnormal returns using the Kolari and Pynnoenen (2010) correction. Abnormal returns are standardized by each event's MODEL sigma (model\$statistics\$sigma, same as [BMPTest](#)) – not the forecast-error-corrected sigma. The adjustment scales the BMP statistic by a factor that accounts for the average pairwise correlation of standardized abnormal residuals in the estimation window, computed only over "usable" events (at least 2 finite estimation-window SARs, finite model sigma, and non-constant/non-degenerate SAR variance). The adjustment is never silently substituted with 1: an event that cannot contribute a usable estimation-window series is excluded from the correlation estimate and reported once; if fewer than 2 usable events remain, kp_t and ckp_t are NA.

Super class

[TestStatisticBase](#) -> KolariPynnonenTest

Public fields

name Short code of the test statistic.

Methods

Public methods:

- [KolariPynnonenTest\\$compute\(\)](#)
- [KolariPynnonenTest\\$clone\(\)](#)

KolariPynnonenTest\$compute(): Computes the Kolari-Pynnoenen adjusted BMP test.

Usage:

```
KolariPynnonenTest$compute(data_tbl, model)
```

Arguments:

data_tbl The data for a multiple event with calculated abnormal returns.

model The fitted model containing sigma estimates.

KolariPynnonenTest\$clone(): The objects of this class are cloneable with this method.

Usage:

```
KolariPynnonenTest$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

References

Kolari, J. W. and Pynnoenen, S. (2010). Event Study Testing with Cross-sectional Correlation of Abnormal Returns. *The Review of Financial Studies*, 23(11), 3996–4025.

See Also

Other eventstudy-statistics: [ARTTest](#), [BHARTTest](#), [BMPTTest](#), [CARTTest](#), [CSectTTest](#), [CalendarTimePortfolioTest](#), [GeneralizedSignTest](#), [MultiEventStatisticsSet](#), [PatellZTest](#), [RankTest](#), [SignTest](#), [SingleEventStatisticsSet](#), [StatisticsSetBase](#), [TestStatisticBase](#), [adjust_p_values\(\)](#), [bootstrap_test\(\)](#), [car_by_group\(\)](#), [car_quantiles\(\)](#), [cross_sectional_regression\(\)](#), [model_diagnostics\(\)](#), [pretrend_test\(\)](#), [simulate_event_study\(\)](#), [validate_task\(\)](#)

LinearFactorModel

Linear Factor Model Base

Description

Base class for multi-factor OLS models used in event studies. All factor models (Market Model, Fama-French, Carhart) share the same estimation approach: OLS regression of excess returns on factor returns during the estimation window.

Super class

[ModelBase](#) -> LinearFactorModel

Public fields

`model_name` Name of the model.
`formula` The regression formula.
`required_columns` Columns required in the data.
`use_hac` Logical. Use HAC (Newey-West) standard errors.
`hac_lag` Integer or NULL. Lag truncation for Newey-West.

Methods**Public methods:**

- [LinearFactorModel\\$new\(\)](#)
- [LinearFactorModel\\$fit\(\)](#)
- [LinearFactorModel\\$abnormal_returns\(\)](#)
- [LinearFactorModel\\$clone\(\)](#)

`LinearFactorModel$new()`: Create a new LinearFactorModel.

Usage:

```
LinearFactorModel$new(use_hac = FALSE, hac_lag = NULL)
```

Arguments:

`use_hac` Logical. Use HAC (Newey-West) standard errors.
`hac_lag` Integer or NULL. Lag truncation for Newey-West.

`LinearFactorModel$fit()`: Fit the linear factor model via OLS on the estimation window.

Usage:

```
LinearFactorModel$fit(data_tbl)
```

Arguments:

`data_tbl` Data frame or tibble containing the data to fit.

`LinearFactorModel$abnormal_returns()`: Calculate abnormal returns as observed minus predicted.

Usage:

```
LinearFactorModel$abnormal_returns(data_tbl)
```

Arguments:

`data_tbl` Data frame or tibble.

`LinearFactorModel$clone()`: The objects of this class are cloneable with this method.

Usage:

```
LinearFactorModel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-models: [BHARModel](#), [Carhart4FactorModel](#), [ComparisonPeriodMeanAdjustedModel](#), [DCCGARCHModel](#), [FamaFrench3FactorModel](#), [FamaFrench5FactorModel](#), [GARCHModel](#), [LogReturn](#), [MarketAdjustedModel](#), [MarketModel](#), [ModelBase](#), [ReturnCalculation](#), [RollingWindowModel](#), [SimpleReturn](#), [VolatilityModel](#), [VolumeModel](#)

LogReturn

R6 class for log return calculation

Description

R6 class for log return calculation

Super class

[ReturnCalculation](#) -> LogReturn

Public fields

`name` Name of the log return calculation.

Methods**Public methods:**

- [LogReturn\\$calculate_return\(\)](#)
- [LogReturn\\$clone\(\)](#)

[LogReturn\\$calculate_return\(\)](#): Calculates the return for a single stock.

Usage:

```
LogReturn$calculate_return(
  tbl,
  in_column = "adjusted",
  out_column = "adjusted_return"
)
```

Arguments:

tbl The dataframe with the stock price.

in_column The column name of the price information.

out_column The column name where the return will be saved.

[LogReturn\\$clone\(\)](#): The objects of this class are cloneable with this method.

Usage:

```
LogReturn$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other eventstudy-models: [BHARModel](#), [Carhart4FactorModel](#), [ComparisonPeriodMeanAdjustedModel](#), [DCCGARCHModel](#), [FamaFrench3FactorModel](#), [FamaFrench5FactorModel](#), [GARCHModel](#), [LinearFactorModel](#), [MarketAdjustedModel](#), [MarketModel](#), [ModelBase](#), [ReturnCalculation](#), [RollingWindowModel](#), [SimpleReturn](#), [VolatilityModel](#), [VolumeModel](#)

MarketAdjustedModel	<i>Market Adjusted Model</i>
---------------------	------------------------------

Description

The Market Adjusted Model is another simple approach used in event studies to estimate the expected returns of a stock and calculate its abnormal returns during an event window. This model is less complex than the Market Model, as it assumes that a stock's expected return is equal to the market return, without considering any stock-specific factors. The Market Adjusted Model is particularly useful in situations where the estimation of individual stock parameters (such as alpha and beta) is not feasible or desired, and a basic benchmark for comparison is needed.

Super class

[ModelBase](#) -> MarketAdjustedModel

Public fields

model_name Name of the model.

Methods**Public methods:**

- [MarketAdjustedModel\\$fit\(\)](#)
- [MarketAdjustedModel\\$abnormal_returns\(\)](#)
- [MarketAdjustedModel\\$clone\(\)](#)

`MarketAdjustedModel$fit()`: fit Fit the model with given data.

Usage:

```
MarketAdjustedModel$fit(data_tbl)
```

Arguments:

data_tbl Data frame or tibble containing the data to fit.

`MarketAdjustedModel$abnormal_returns()`: abnormal_returns Calculate the abnormal returns with given data.

Usage:

```
MarketAdjustedModel$abnormal_returns(data_tbl)
```

Arguments:

data_tbl Data frame or tibble containing the data to calculate abnormal returns.

`MarketAdjustedModel$clone()`: The objects of this class are cloneable with this method.

Usage:

```
MarketAdjustedModel$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other eventstudy-models: [BHARModel](#), [Carhart4FactorModel](#), [ComparisonPeriodMeanAdjustedModel](#), [DCCGARCHModel](#), [FamaFrench3FactorModel](#), [FamaFrench5FactorModel](#), [GARCHModel](#), [LinearFactorModel](#), [LogReturn](#), [MarketModel](#), [ModelBase](#), [ReturnCalculation](#), [RollingWindowModel](#), [SimpleReturn](#), [VolatilityModel](#), [VolumeModel](#)

MarketModel

Market Model

Description

The Market Model is a widely used method in event studies to estimate the expected returns of a stock and calculate its abnormal returns during an event window. The model is based on a simple linear regression framework and captures the relationship between a stock's return and the return of a market index, such as the S&P 500 or the Dow Jones Industrial Average. The underlying assumption of the Market Model is that a stock's return is primarily influenced by market movements, along with a stock-specific idiosyncratic component.

Super class

`ModelBase` -> `MarketModel`

Public fields

`model_name` Name of the model.

`formula` The formula applied for calculating the market model

`use_hac` Logical. Use HAC (Newey-West) standard errors.

`hac_lag` Integer or NULL. Lag truncation for Newey-West. NULL uses the automatic bandwidth selection.

Methods

Public methods:

- `MarketModel$new()`
- `MarketModel$set_formula()`
- `MarketModel$fit()`
- `MarketModel$abnormal_returns()`
- `MarketModel$clone()`

`MarketModel$new()`: Create a new `MarketModel`.

Usage:

```
MarketModel$new(use_hac = FALSE, hac_lag = NULL)
```

Arguments:

`use_hac` Logical. Use HAC (Newey-West) standard errors. Requires the **sandwich** package.

`hac_lag` Integer or NULL. Lag truncation for Newey-West.

`MarketModel$set_formula()`: Set the formula

Usage:

```
MarketModel$set_formula(formula)
```

Arguments:

formula A formula.

`MarketModel$fit()`: Fit the model with given data.

Usage:

`MarketModel$fit(data_tbl)`

Arguments:

`data_tbl` Data frame or tibble containing the data to fit.

`MarketModel$abnormal_returns()`: Calculate the abnormal returns with given data.

Usage:

`MarketModel$abnormal_returns(data_tbl)`

Arguments:

`data_tbl` Data frame or tibble containing the data to calculate abnormal returns.

`MarketModel$clone()`: The objects of this class are cloneable with this method.

Usage:

`MarketModel$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-models: [BHARModel](#), [Carhart4FactorModel](#), [ComparisonPeriodMeanAdjustedModel](#), [DCCGARCHModel](#), [FamaFrench3FactorModel](#), [FamaFrench5FactorModel](#), [GARCHModel](#), [LinearFactorModel](#), [LogReturn](#), [MarketAdjustedModel](#), [ModelBase](#), [ReturnCalculation](#), [RollingWindowModel](#), [SimpleReturn](#), [VolatilityModel](#), [VolumeModel](#)

ModelBase

ModelBase

Description

Base (abstract) class for return models used inside an `EventStudyTask`. Each single event gets its own model instance, fitted on that event's estimation-window data alone. Subclass this to plug a custom return model into `run_event_study()` / `fit_model()`.

A subclass must implement two methods:

- `fit(data_tbl)` – fit the model on `data_tbl` (a single event's rows, estimation and event window together) and populate `private$.statistics` plus set `private$.is_fitted`.
- `abnormal_returns(data_tbl)` – return `data_tbl` with an `abnormal_returns` column added for the event window.

The statistics active binding exposes a named list consumed by the test statistics layer. At minimum a subclass should populate: `sigma`, `degree_of_freedom`, `residuals`, `forecast_error_corrected_sigma` and `forecast_error_corrected_sigma_car`. Since 2026-09-24, a model may also set `statistics$n_params` (the number of parameters estimated from the estimation window); `PatellZTest` reads this field for its `Q_i` adjustment and falls back to 2 when a subclass does not report it.

Degenerate-input handling: subclasses that implement the shared contract (see `R/contract.R`) read `self$degenerate_mode`, `self$event_id` and `self$firm_symbol`, which are threaded onto the cloned model instance by `fit_model()` before `fit()` is called.

Public fields

`model_name` Name of the model.

`degenerate_mode` Resolved degenerate-input mode injected by `fit_model()` before `fit()` is called. Subclasses that implement the degenerate-input contract (`MarketModel` and `Phase-2` models) read this field inside `fit()` via `.resolve_degenerate_mode(self$degenerate_mode)`.

`event_id` Event identifier threaded from the outer `data_tbl` row. Used in degenerate-input error/warning messages.

`firm_symbol` Firm identifier threaded from the outer `data_tbl` row. Used in degenerate-input error/warning messages.

Active bindings

`statistics` Read-only field to get statistics.

`model` Read-only field to get the fitted model.

`is_fitted` Read-only field to check if the model is fitted. Statistics object contains different model specific KPIs that describes the fitted model.

Methods

Public methods:

- `ModelBase$fit()`
- `ModelBase$abnormal_returns()`
- `ModelBase$clone()`

`ModelBase$fit()`: Fits the model with given data.

Usage:

```
ModelBase$fit(data_tbl)
```

Arguments:

`data_tbl` A data frame or tibble containing the data to fit.

`ModelBase$abnormal_returns()`: Calculate the abnormal returns with given data and fitted model.

Usage:

```
ModelBase$abnormal_returns(data_tbl)
```

Arguments:

`data_tbl` Data frame or tibble containing the data to calculate abnormal returns.

`ModelBase$clone()`: The objects of this class are cloneable with this method.

Usage:

`ModelBase$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-models: [BHARModel](#), [Carhart4FactorModel](#), [ComparisonPeriodMeanAdjustedModel](#), [DCCGARCHModel](#), [FamaFrench3FactorModel](#), [FamaFrench5FactorModel](#), [GARCHModel](#), [LinearFactorModel](#), [LogReturn](#), [MarketAdjustedModel](#), [MarketModel](#), [ReturnCalculation](#), [RollingWindowModel](#), [SimpleReturn](#), [VolatilityModel](#), [VolumeModel](#)

model_diagnostics	<i>Model Diagnostics for Event Study</i>
-------------------	--

Description

Run diagnostic tests on fitted event study models. Tests include normality of residuals, autocorrelation, and basic model fit quality.

Usage

`model_diagnostics(task, event_id = NULL)`

Arguments

<code>task</code>	A fitted <code>EventStudyTask</code> (after <code>fit_model</code> has been called).
<code>event_id</code>	Optional event identifier. If <code>NULL</code> , runs diagnostics for all events.

Value

A tibble with diagnostic test results for each event.

See Also

Other eventstudy-statistics: [ARTTest](#), [BHARTTest](#), [BMPTTest](#), [CARTTest](#), [CSectTTest](#), [CalendarTimePortfolioTest](#), [GeneralizedSignTest](#), [KolariPynnonenTest](#), [MultiEventStatisticsSet](#), [PatellZTest](#), [RankTest](#), [SignTest](#), [SingleEventStatisticsSet](#), [StatisticsSetBase](#), [TestStatisticBase](#), [adjust_p_values\(\)](#), [bootstrap_test\(\)](#), [car_by_group\(\)](#), [car_quantiles\(\)](#), [cross_sectional_regression\(\)](#), [pretrend_test\(\)](#), [simulate_event_study\(\)](#), [validate_task\(\)](#)

`MultiEventStatisticsSet`*Multi Event Statistic Set*

Description

Contains a set of multi event test statistics as, e.g., AAR and CAAR cross-sectional t test. These are the default test statistics.

Super class

`StatisticsSetBase` -> `MultiEventStatisticsSet`

Public fields

`tests` Container with the statistical tests.

Methods

Public methods:

- `MultiEventStatisticsSet$clone()`

`MultiEventStatisticsSet$clone()`: The objects of this class are cloneable with this method.

Usage:

`MultiEventStatisticsSet$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-statistics: `ARTTest`, `BHARTTest`, `BMPTTest`, `CARTTest`, `CSectTTest`, `CalendarTimePortfolioTest`, `GeneralizedSignTest`, `KolariPynnonenTest`, `PatellZTest`, `RankTest`, `SignTest`, `SingleEventStatisticsSet`, `StatisticsSetBase`, `TestStatisticBase`, `adjust_p_values()`, `bootstrap_test()`, `car_by_group()`, `car_quantiles()`, `cross_sectional_regression()`, `model_diagnostics()`, `pretrend_test()`, `simulate_event_study()`, `validate_task()`

nonparametric_intraday_test

Non-Parametric Intraday Event Study Test

Description

Implements the non-parametric intraday event study methodology of Rinaudo & Saha (2014). For each event time, cumulative abnormal returns (CARs) on the event day are compared to the empirical distribution of CARs from estimation-period days. Significance is assessed without distributional assumptions.

Usage

```
nonparametric_intraday_test(
  estimation_window,
  event_window,
  event_times,
  p = 0.05,
  init_window = 5L,
  upper = FALSE
)
```

Arguments

estimation_window	Data frame with columns day, time, and abnormalReturn containing intraday abnormal returns for multiple estimation days.
event_window	Data frame with columns time and abnormalReturn for the single event day.
event_times	Character vector of intraday times at which events occurred (e.g., c("10:00", "14:30")).
p	Numeric; significance level in (0, 1). Default 0.05.
init_window	Integer; number of initial observations to accumulate before testing significance. Default 5.
upper	Logical; if TRUE test for positive abnormal returns (upper tail), otherwise test for negative abnormal returns (lower tail). Default FALSE.

Value

A named list of tibbles, one per event time. Each tibble contains:

id Observation index within the significant window.

CAR Cumulative abnormal return on the event day.

CI Empirical confidence interval boundary from estimation days.

fitCI Polynomial-smoothed CI (degree-4 polynomial fit).

When an event time is not significant, returns a single-row tibble with all values equal to zero.

References

Rinaudo, J.B. & Saha, A. (2014). Non-parametric intraday event studies.

See Also

Other eventstudy-tasks: [IntradayEventStudyTask](#), [PanelEventStudyTask](#), [SyntheticControlTask](#), [estimate_panel_event_study\(\)](#), [estimate_synthetic_control\(\)](#), [plot_panel_event_study\(\)](#), [plot_synthetic_control\(\)](#), [prepare_intraday_event_study\(\)](#), [sc_placebo_test\(\)](#)

Examples

```
# Build a small intraday grid: estimation window (multiple days) and a
# single event-day window sharing the same intraday time stamps.
set.seed(1)
times <- sprintf("%02d:%02d", 10L + seq_len(30L) %% 60L, seq_len(30L) %% 60L)
est_data <- data.frame(
  day          = rep(seq_len(10L), each = 30L),
  time         = rep(times, 10L),
  abnormalReturn = rnorm(300L, 0, 0.001)
)
event_data <- data.frame(
  time         = times,
  abnormalReturn = rnorm(30L, 0, 0.001)
)

results <- nonparametric_intraday_test(
  estimation_window = est_data,
  event_window      = event_data,
  event_times       = times[c(5L, 20L)],
  p                 = 0.05,
  init_window       = 5
)
```

OpenAICompatProvider *OpenAICompatProvider*

Description

A provider that issues POST {base_url}/chat/completions with an OpenAI-shaped request body. Because the endpoint is selected purely by base_url + model, this single class covers OpenAI itself AND every OpenAI-compatible backend – Ollama, LM Studio, or any gateway – via a base_url override, with no code change.

The API key is resolved at CALL time from OPENAI_API_KEY (never at construction, never stored on the object) and attached with req_auth_bearer_token(), which redacts the Authorization header in every print/error path. A missing key, transport/timeout failure, non-2xx status, malformed body, or missing completion text each degrades to exactly one warning() plus an ‘es_provider_response’

with `text = NA_character_` – it never crashes the session and never returns a fabricated completion.

`httr2` is required only for this provider and is guarded by `requireNamespace()` at the top of `complete()`, so the package installs and R CMD checks cleanly with `httr2` absent.

Super class

`ProviderBase` -> `OpenAICompatProvider`

Methods

Public methods:

- `OpenAICompatProvider$new()`
- `OpenAICompatProvider$complete()`
- `OpenAICompatProvider$clone()`

`OpenAICompatProvider$new()`: Construct an `OpenAICompatProvider`. Stores non-secret config only; the API key is resolved at call time inside `complete()`.

Usage:

```
OpenAICompatProvider$new(model, base_url = "https://api.openai.com/v1", ...)
```

Arguments:

`model` Character model identifier, e.g. "gpt-4o" or a local model name like "llama3".

`base_url` Character base URL. Defaults to the OpenAI cloud endpoint; override for OpenAI-compatible servers (Ollama, LM Studio, gateways).

... Ignored; accepted for forward-compatibility.

`OpenAICompatProvider$complete()`: Complete a prompt via POST `{base_url}/chat/completions`. Resolves the key at call time; on a missing key or any HTTP/parse failure returns one warning + NA. Structured output is OPTIONAL: when schema is supplied a `response_format json_schema` is added to the body, but omitting it uses the plain-text path (which local models that lack `response_format` support still handle).

Usage:

```
OpenAICompatProvider$complete(prompt, schema = NULL, ...)
```

Arguments:

`prompt` Character prompt.

`schema` Optional structured-output schema (list) or NULL.

... Ignored; accepted for forward-compatibility.

Returns: An 'es_provider_response' (see `ProviderBase$complete`).

`OpenAICompatProvider$clone()`: The objects of this class are cloneable with this method.

Usage:

```
OpenAICompatProvider$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-advisor: [AnthropicProvider](#), [CustomProvider](#), [ProviderBase](#), [es_advise\(\)](#), [es_diagnostics\(\)](#), [es_kb\(\)](#), [flag_robustness\(\)](#), [provider\(\)](#), [recommend_stat\(\)](#)

Examples

```
## Not run:
# OpenAI (reads OPENAI_API_KEY from the environment at call time):
p <- OpenAICompatProvider$new(model = "gpt-4o")
p$complete("Summarise these event-study diagnostics")

# Any OpenAI-compatible endpoint works via a base_url override -- e.g. a local
# Ollama server (no cloud key needed by the server, but OPENAI_API_KEY is still
# read as the bearer token; set it to any non-empty value for local servers):
p_local <- OpenAICompatProvider$new(
  model = "llama3",
  base_url = "http://localhost:11434/v1" # Ollama; LM Studio: :1234/v1
)
p_local$complete("Summarise these diagnostics")

## End(Not run)
```

PanelEventStudyTask *Panel Event Study Task*

Description

Task container for panel (difference-in-differences style) event studies. This is designed for settings where units receive treatment at potentially different times, and we observe outcomes over multiple periods.

Methods

`new(panel_data, unit_id, time_id, outcome, treatment, treatment_time)` Create a new panel event study task.

`print()` Print summary.

Public fields

`panel_data` The panel dataset in long format.

`unit_id` Name of the unit identifier column.

`time_id` Name of the time identifier column.

`outcome` Name of the outcome column.

`treatment` Name of the treatment indicator column (0/1).

`treatment_time` Name of the treatment timing column (period when unit first treated; NA for never-treated).

`results` Estimation results (populated after estimation).

Methods

Public methods:

- [PanelEventStudyTask\\$new\(\)](#)
- [PanelEventStudyTask\\$print\(\)](#)
- [PanelEventStudyTask\\$clone\(\)](#)

`PanelEventStudyTask$new()`: Create a new `PanelEventStudyTask`.

Usage:

```
PanelEventStudyTask$new(
  panel_data,
  unit_id = "unit_id",
  time_id = "time_id",
  outcome = "outcome",
  treatment = "treated",
  treatment_time = "treatment_time"
)
```

Arguments:

`panel_data` Long-format panel data frame.

`unit_id` Name of the unit ID column.

`time_id` Name of the time period column.

`outcome` Name of the outcome variable column.

`treatment` Name of the treatment indicator column (0/1).

`treatment_time` Name of column indicating when each unit is first treated (NA for never-treated controls).

`PanelEventStudyTask$print()`: Print summary.

Usage:

```
PanelEventStudyTask$print(...)
```

`PanelEventStudyTask$clone()`: The objects of this class are cloneable with this method.

Usage:

```
PanelEventStudyTask$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-tasks: [IntradayEventStudyTask](#), [SyntheticControlTask](#), [estimate_panel_event_study\(\)](#), [estimate_synthetic_control\(\)](#), [nonparametric_intraday_test\(\)](#), [plot_panel_event_study\(\)](#), [plot_synthetic_control\(\)](#), [prepare_intraday_event_study\(\)](#), [sc_placebo_test\(\)](#)

ParameterSet	<i>Event Study Parameter Set</i>
--------------	----------------------------------

Description

The parameter set defines the Event Study, e.g, the return calculation, the event study model, the AR, CAR, AAR, and CAAR test statistics that should be applied.

Public fields

`return_calculation` A R6 object for calculating returns.

`return_model` A R6 object for fitting the desired model.

`single_event_statistics` single event test statistic R6 object.

`multi_event_statistics` multi event test statistic R6 object.

`study_type` Type of event study: "return" (default), "volume", or "volatility". Affects axis labels in plots.

`degenerate_handling` Controls how degenerate estimation data is handled. One of "lenient" (default), "strict", or NULL (defer to package option or built-in default). See `?degenerate-input-contract` for details.

Methods

Public methods:

- [ParameterSet\\$new\(\)](#)
- [ParameterSet\\$print\(\)](#)
- [ParameterSet\\$clone\(\)](#)

`ParameterSet$new()`: Initialize the parameters that defines the Event Study that should be applied.

Usage:

```
ParameterSet$new(
  return_calculation = SimpleReturn$new(),
  return_model = MarketModel$new(),
  single_event_statistics = SingleEventStatisticsSet$new(),
  multi_event_statistics = MultiEventStatisticsSet$new(),
  degenerate_handling = NULL
)
```

Arguments:

`return_calculation` An initialized return calculation class. Defaults to `SimpleReturn`.

`return_model` An initialized event study model. Defaults to `MarketModel`.

`single_event_statistics` Definition of single event test statistics. Defaults to `SingleEventStatisticsSet` (AR T and CAR T tests).

`multi_event_statistics` Definition of multiple event test statistics. Defaults to `MultiEventStatisticsSet` (CSect T test).

degenerate_handling One of "lenient", "strict", or NULL. See ?degenerate-input-contract.

ParameterSet\$print(): Print a summary of the parameter set.

Usage:

ParameterSet\$print(...)

ParameterSet\$clone(): The objects of this class are cloneable with this method.

Usage:

ParameterSet\$clone(deep = FALSE)

Arguments:

deep Whether to make a deep clone.

See Also

Other eventstudy-pipeline: [EventStudyTask](#), [calculate_statistics\(\)](#), [fit_model\(\)](#), [prepare_event_study\(\)](#), [run_event_study\(\)](#)

PatellZTest

Patell or Standardized Residual Test (PatellZTest)

Description

The Patell or Standardized Residual Test is a statistical tool employed in event studies to evaluate the null hypothesis that the average abnormal return at the event date is zero. The test statistics is approximately distributed as $N(0, 1)$.

See also https://eventstudy.de/statistics/aar_caar_statistics.html

Super class

[TestStatisticBase](#) -> PatellZTest

Public fields

name Short code of the test statistic.

Methods

Public methods:

- [PatellZTest\\$compute\(\)](#)
- [PatellZTest\\$clone\(\)](#)

PatellZTest\$compute(): Computes the Patell Z test statistics for multiple events.

Usage:

PatellZTest\$compute(data_tbl, model)

Arguments:

`data_tbl` The data for a multiple event with calculated abnormal returns.
`model` The fitted model.

`PatellZTest$clone()`: The objects of this class are cloneable with this method.

Usage:

`PatellZTest$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-statistics: [ARTTest](#), [BHARTTest](#), [BMPTTest](#), [CARTTest](#), [CSectTTest](#), [CalendarTimePortfolioTest](#), [GeneralizedSignTest](#), [KolariPynnonenTest](#), [MultiEventStatisticsSet](#), [RankTest](#), [SignTest](#), [SingleEventStatisticsSet](#), [StatisticsSetBase](#), [TestStatisticBase](#), [adjust_p_values\(\)](#), [bootstrap_test\(\)](#), [car_by_group\(\)](#), [car_quantiles\(\)](#), [cross_sectional_regression\(\)](#), [model_diagnostics\(\)](#), [pretrend_test\(\)](#), [simulate_event_study\(\)](#), [validate_task\(\)](#)

`plot_car_distribution` *Plot CAR Distribution*

Description

Create a histogram of CARs across events, optionally colored by group.

Usage

```
plot_car_distribution(  
  task,  
  car_window = NULL,  
  bins = 30,  
  by_group = FALSE,  
  title = NULL  
)
```

Arguments

<code>task</code>	A fitted <code>EventStudyTask</code> .
<code>car_window</code>	Optional two-element vector for CAR window.
<code>bins</code>	Number of histogram bins. Default 30.
<code>by_group</code>	Logical. If TRUE, color histogram by group.
<code>title</code>	Optional plot title.

Value

A `ggplot2` object.

See Also

Other eventstudy-plots: [es_colours](#), [plot_diagnostics\(\)](#), [plot_event_study\(\)](#), [plot_stocks\(\)](#), [theme_eventstudy\(\)](#)

plot_diagnostics	<i>Plot Model Diagnostics</i>
------------------	-------------------------------

Description

Create diagnostic plots for fitted event study models, including residual plots and Q-Q plots.

Usage

```
plot_diagnostics(task, event_id = NULL)
```

Arguments

- task A fitted EventStudyTask.
- event_id The event identifier to plot diagnostics for.

Value

A ggplot2 plot arranged in a gridExtra 2x2 grid layout.

See Also

Other eventstudy-plots: [es_colours](#), [plot_car_distribution\(\)](#), [plot_event_study\(\)](#), [plot_stocks\(\)](#), [theme_eventstudy\(\)](#)

plot_event_study	<i>Plot Event Study Results</i>
------------------	---------------------------------

Description

Create an event study plot showing abnormal returns (AR), cumulative abnormal returns (CAR), average abnormal returns (AAR), or cumulative average abnormal returns (CAAR) with confidence bands.

Usage

```
plot_event_study(
  task,
  type = "car",
  event_id = NULL,
  group = NULL,
  stat_name = "CSectT",
  confidence_level = 0.95,
  title = NULL
)
```

Arguments

task	A fitted EventStudyTask with statistics computed.
type	Type of plot: "ar", "car", "aar", or "caar".
event_id	Event identifier for single-event plots ("ar", "car").
group	Group name for multi-event plots ("aar", "caar").
stat_name	Name of the multi-event test statistic to use for AAR/CAAR plots. Defaults to "CSectT".
confidence_level	Confidence level for the bands. Default is 0.95.
title	Optional plot title.

Value

A ggplot2 plot object.

See Also

Other eventstudy-plots: [es_colours](#), [plot_car_distribution\(\)](#), [plot_diagnostics\(\)](#), [plot_stocks\(\)](#), [theme_eventstudy\(\)](#)

plot_panel_event_study

Plot Panel Event Study Results

Description

Create an event study plot from panel estimation results. Can also be called via `plot_event_study()` when given a `PanelEventStudyTask`.

Usage

```
plot_panel_event_study(task, confidence_level = 0.95, title = NULL)
```

Arguments

task	A PanelEventStudyTask with results.
confidence_level	Confidence level for error bars. Default 0.95.
title	Optional plot title.

Value

A ggplot2 object.

See Also

Other eventstudy-tasks: [IntradayEventStudyTask](#), [PanelEventStudyTask](#), [SyntheticControlTask](#), [estimate_panel_event_study\(\)](#), [estimate_synthetic_control\(\)](#), [nonparametric_intraday_test\(\)](#), [plot_synthetic_control\(\)](#), [prepare_intraday_event_study\(\)](#), [sc_placebo_test\(\)](#)

plot_stocks	<i>Plot Stocks</i>
-------------	--------------------

Description

Visualize the adjusted close prices or abnormal returns of selected stocks around a specified event date.

Usage

```
plot_stocks(  
  task,  
  target_variable = "firm_adjusted",  
  add_event_date = FALSE,  
  max_symbols = 6,  
  sample_symbols = TRUE,  
  do_sample = NULL  
)
```

Arguments

task	An EventStudyTask object.
target_variable	A character string specifying the target variable to plot. Default is "firm_adjusted".
add_event_date	Add vertical line for event date (TRUE/FALSE).
max_symbols	An integer specifying the maximum number of symbols to display. Default is 6.
sample_symbols	Logical; if TRUE (default), randomly sample max_symbols symbols when the task contains more than max_symbols. If FALSE, take the first max_symbols in order.
do_sample	<i>Deprecated.</i> Renamed to sample_symbols in EventStudy 0.66.0. Accepts the old name for backward compatibility and emits one deprecation warning.

Value

A plotly plot object.

See Also

Other eventstudy-plots: [es_colours](#), [plot_car_distribution\(\)](#), [plot_diagnostics\(\)](#), [plot_event_study\(\)](#), [theme_eventstudy\(\)](#)

plot_synthetic_control

Plot Synthetic Control Results

Description

Create plots for synthetic control analysis: trajectory comparison, treatment effect gap, or placebo test results.

Usage

```
plot_synthetic_control(task, type = c("trajectory", "gap", "placebo"))
```

Arguments

task	A SyntheticControlTask with results.
type	Plot type: "trajectory" (default), "gap", or "placebo".

Value

A ggplot2 object.

See Also

Other eventstudy-tasks: [IntradayEventStudyTask](#), [PanelEventStudyTask](#), [SyntheticControlTask](#), [estimate_panel_event_study\(\)](#), [estimate_synthetic_control\(\)](#), [nonparametric_intraday_test\(\)](#), [plot_panel_event_study\(\)](#), [prepare_intraday_event_study\(\)](#), [sc_placebo_test\(\)](#)

prepare_event_study	<i>Prepare data for an Event Study</i>
---------------------	--

Description

Perform return calculation for each stock and the corresponding reference market defined in the task and the parameter set.

Usage

```
prepare_event_study(task, parameter_set)
```

Arguments

task	An Event Study task.
parameter_set	A parameter set that defines the Event Study.

Value

The task object with returns and windows appended.

See Also

[run_event_study](#), [fit_model](#), [calculate_statistics](#)

Other eventstudy-pipeline: [EventStudyTask](#), [ParameterSet](#), [calculate_statistics\(\)](#), [fit_model\(\)](#), [run_event_study\(\)](#)

prepare_intraday_event_study	<i>Prepare Intraday Event Study</i>
------------------------------	-------------------------------------

Description

Compute returns and assign estimation/event windows for intraday data. Windows are specified in number of observations (bars) rather than days.

Usage

```
prepare_intraday_event_study(task, parameter_set)
```

Arguments

task	An IntradayEventStudyTask.
parameter_set	A ParameterSet.

Value

The task with returns and windows appended.

See Also

Other eventstudy-tasks: [IntradayEventStudyTask](#), [PanelEventStudyTask](#), [SyntheticControlTask](#), [estimate_panel_event_study\(\)](#), [estimate_synthetic_control\(\)](#), [nonparametric_intraday_test\(\)](#), [plot_panel_event_study\(\)](#), [plot_synthetic_control\(\)](#), [sc_placebo_test\(\)](#)

pretrend_test	<i>Pre-trend Test for Event Study</i>
---------------	---------------------------------------

Description

Performs a joint F-test of pre-event abnormal returns to assess whether there are statistically significant pre-event effects, which would indicate potential model misspecification or confounding.

Usage

```
pretrend_test(task, group = NULL)
```

Arguments

task	A fitted EventStudyTask with abnormal returns computed.
group	Optional group to filter by.

Value

A tibble with pre-trend test results for each group.

See Also

Other eventstudy-statistics: [ARTTest](#), [BHARTTest](#), [BMPTTest](#), [CARTTest](#), [CSectTTest](#), [CalendarTimePortfolioTest](#), [GeneralizedSignTest](#), [KolariPynnonenTest](#), [MultiEventStatisticsSet](#), [PatellZTest](#), [RankTest](#), [SignTest](#), [SingleEventStatisticsSet](#), [StatisticsSetBase](#), [TestStatisticBase](#), [adjust_p_values\(\)](#), [bootstrap_test\(\)](#), [car_by_group\(\)](#), [car_quantiles\(\)](#), [cross_sectional_regression\(\)](#), [model_diagnostics\(\)](#), [simulate_event_study\(\)](#), [validate_task\(\)](#)

<code>print.Advice</code>	<i>Print method for Advice objects</i>
---------------------------	--

Description

Prints a structured summary of the grounded AI advice, including source, task type, grounding guard status, interpretation, recommendations with evidence, and caveats. Follows the package convention of `print.es_advice` (delegates to `format.Advice`, invisible return).

Usage

```
## S3 method for class 'Advice'
print(x, ...)
```

Arguments

- `x` An object of class "Advice".
- `...` Additional arguments (ignored).

Value

Invisibly returns `x`.

<code>print.es_advice</code>	<i>Print method for es_advice objects</i>
------------------------------	---

Description

Prints a structured summary of the offline advice, listing each matched rule with its severity, citation key, and recommendation. Follows the package convention of `print.es_diagnostics` and `print.es_simulation` (delegates to `format.es_advice`, invisible return).

Usage

```
## S3 method for class 'es_advice'
print(x, ...)
```

Arguments

- `x` An object of class "es_advice".
- `...` Additional arguments (ignored).

Value

Invisibly returns `x`.

```
print.es_cross_sectional
```

Print Cross-Sectional Regression Results

Description

Print Cross-Sectional Regression Results

Usage

```
## S3 method for class 'es_cross_sectional'  
print(x, ...)
```

Arguments

x	An es_cross_sectional object.
...	Additional arguments (unused).

```
print.es_diagnostics
```

Print method for es_diagnostics objects

Description

Prints a concise summary of the diagnostics, following the package convention of `print.es_simulation` and `print.es_cross_sectional`.

Usage

```
## S3 method for class 'es_diagnostics'  
print(x, ...)
```

Arguments

x	An object of class "es_diagnostics".
...	Additional arguments (ignored).

Value

Invisibly returns x.

<code>print.EventStudySummary</code>	<i>Print method for EventStudySummary</i>
--------------------------------------	---

Description

Print method for EventStudySummary

Usage

```
## S3 method for class 'EventStudySummary'
print(x, ...)
```

Arguments

- `x` An EventStudySummary object.
- `...` Additional arguments (unused).

<code>provider</code>	<i>Construct a Grounded AI Advisor provider</i>
-----------------------	---

Description

Convenience factory that resolves the backend by 3-tier precedence (explicit type argument -> EVENTSTUDY_ADVISOR_PROVIDER env selector -> default "custom") and constructs the matching provider object. All three backends are wired: the in-process "custom" hook and the HTTP "openai" (OpenAI-compatible) and "anthropic" (native Anthropic Messages) providers.

Usage

```
provider(type = NULL, fn = NULL, model = NULL, base_url = NULL, ...)
```

Arguments

- `type` Provider type, one of "custom", "openai", "anthropic". When NULL (the default) it is resolved from the EVENTSTUDY_ADVISOR_PROVIDER env var, falling back to "custom".
- `fn` For type = "custom", the user function function(prompt, schema, ...) returning a list or character. Required for the custom branch.
- `model` Optional character model identifier (resolved from EVENTSTUDY_ADVISOR_MODEL when NULL).
- `base_url` Optional character base URL for HTTP providers (resolved from EVENTSTUDY_ADVISOR_BASE_URL when NULL).
- `...` Additional arguments forwarded to the provider constructor.

Details

The "custom" branch needs no network and neither `httr2` nor `jsonlite`. Keys are never read here; the HTTP providers resolve them at call time inside their own `complete()`.

Value

A `ProviderBase` subclass instance.

See Also

Other `eventstudy-advisor`: [AnthropicProvider](#), [CustomProvider](#), [OpenAICompatProvider](#), [ProviderBase](#), [es_advise\(\)](#), [es_diagnostics\(\)](#), [es_kb\(\)](#), [flag_robustness\(\)](#), [recommend_stat\(\)](#)

Examples

```
# Custom provider runs in-process, no network:
p <- provider("custom", fn = function(prompt, schema) "advice text")
p$complete("Summarise")$text

## Not run:
# HTTP providers make live calls; never run in examples or on CRAN:
p <- provider("openai", model = "gpt-4o")
p$complete("Summarise these diagnostics")

a <- provider("anthropic", model = "claude-opus-4-8")
a$complete("Summarise these diagnostics")

## End(Not run)
```

ProviderBase

ProviderBase

Description

Abstract base class for Grounded AI Advisor providers. Defines the uniform `complete(prompt, schema, ...)` contract, mirroring the package's `ModelBase` / `TestStatisticBase` strategy-pattern idiom. Concrete providers (`CustomProvider`, and the HTTP `OpenAICompatProvider` / `AnthropicProvider` delivered in later plans) inherit from this base and implement `complete()`.

`ProviderBase` itself is abstract: calling `complete()` on it errors. It stores only non-secret configuration (`model`, `base_url`); API keys are NEVER read at construction – they are resolved at call time inside each concrete provider's `complete()`.

Public fields

`model` Character model identifier (or NULL for provider default).
`base_url` Character base URL for HTTP providers (or NULL).

Methods

Public methods:

- [ProviderBase\\$new\(\)](#)
- [ProviderBase\\$complete\(\)](#)
- [ProviderBase\\$clone\(\)](#)

ProviderBase\$new(): Construct a provider. Stores non-secret config only; never reads or stores API keys (keys are resolved at call time).

Usage:

```
ProviderBase$new(model = NULL, base_url = NULL, ...)
```

Arguments:

`model` Optional character model identifier.
`base_url` Optional character base URL (HTTP providers only).
`...` Ignored; accepted for subclass forward-compatibility.

ProviderBase\$complete(): Complete a prompt. Abstract on the base class – concrete providers override this.

Usage:

```
ProviderBase$complete(prompt, schema = NULL, ...)
```

Arguments:

`prompt` Character prompt to send to the provider.
`schema` Optional structured-output schema (list) or NULL.
`...` Provider-specific arguments.

Returns: An ‘es_provider_response’ S3 list with fields: `source` (character provider label), `is_deterministic` (always FALSE for provider output), `text` (character completion, or NA_character_ on failure), `error` (NULL on success, character reason on failure). These field names match the Phase 5 `es_advice` shape so the Phase 7 wrapper slots in trivially.

ProviderBase\$clone(): The objects of this class are cloneable with this method.

Usage:

```
ProviderBase$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-advisor: [AnthropicProvider](#), [CustomProvider](#), [OpenAICompatProvider](#), [es_advise\(\)](#), [es_diagnostics\(\)](#), [es_kb\(\)](#), [flag_robustness\(\)](#), [provider\(\)](#), [recommend_stat\(\)](#)

RankTest

*Rank Test (Corrado 1989)***Description**

Non-parametric rank test for event studies. Ranks abnormal returns across the combined estimation and event windows, which is robust to non-normality of abnormal returns.

Super class

[TestStatisticBase](#) -> RankTest

Public fields

name Short code of the test statistic.

Methods**Public methods:**

- [RankTest\\$compute\(\)](#)
- [RankTest\\$clone\(\)](#)

[RankTest\\$compute\(\)](#): Computes the Corrado rank test for multiple events.

Usage:

```
RankTest$compute(data_tbl, model)
```

Arguments:

data_tbl The data for a multiple event with calculated abnormal returns.

model The fitted model (unused).

[RankTest\\$clone\(\)](#): The objects of this class are cloneable with this method.

Usage:

```
RankTest$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other eventstudy-statistics: [ARTTest](#), [BHARTTest](#), [BMPTTest](#), [CARTTest](#), [CSectTTest](#), [CalendarTimePortfolioTest](#), [GeneralizedSignTest](#), [KolariPynnonenTest](#), [MultiEventStatisticsSet](#), [PatellZTest](#), [SignTest](#), [SingleEventStatisticsSet](#), [StatisticsSetBase](#), [TestStatisticBase](#), [adjust_p_values\(\)](#), [bootstrap_test\(\)](#), [car_by_group\(\)](#), [car_quantiles\(\)](#), [cross_sectional_regression\(\)](#), [model_diagnostics\(\)](#), [pretrend_test\(\)](#), [simulate_event_study\(\)](#), [validate_task\(\)](#)

recommend_stat	<i>Recommend Test Statistics via Offline KB Matching</i>
----------------	--

Description

Evaluates the KB decision table (category "stat_choice" rules) against the diagnostic signals extracted from a fitted EventStudyTask or a precomputed es_diagnostics object, and returns a severity-ranked es_advice S3 object.

Usage

```
recommend_stat(x, provider = NULL, ...)

## Default S3 method:
recommend_stat(x, provider = NULL, ...)

## S3 method for class 'EventStudyTask'
recommend_stat(x, provider = NULL, ...)

## S3 method for class 'es_diagnostics'
recommend_stat(x, provider = NULL, ...)
```

Arguments

x	A fitted EventStudyTask (after fit_model()) or a precomputed es_diagnostics object returned by es_diagnostics().
provider	Accepted but ignored in the offline path – present only so the Phase 7 call shape is forward-compatible. Default NULL.
...	Additional arguments (currently ignored).

Details

No LLM provider, network connection, or API key is required. Both functions are the always-available offline grounding layer (ADV-08). The returned es_advice object has the same shape as the Phase 7 Advice contract, flagged is_deterministic = TRUE and source = "offline_kb".

Value

An S3 object of class "es_advice" – a named list with:

source	"offline_kb" (character).
is_deterministic	TRUE – advice is rule-based, not LLM-generated.
rules_matched	Named list of matched rule records (severity-ranked: "error" first, then "warning", then "info"), each with fields id, recommendation, citation (list of author/year/key/venue), severity, category.
diagnostics_ref	The es_diagnostics list that was evaluated (possibly computed on-the-fly from the task).

See Also

[flag_robustness](#), [es_diagnostics](#), [es_kb](#)

Other eventstudy-advisor: [AnthropicProvider](#), [CustomProvider](#), [OpenAICompatProvider](#), [ProviderBase](#), [es_advise\(\)](#), [es_diagnostics\(\)](#), [es_kb\(\)](#), [flag_robustness\(\)](#), [provider\(\)](#)

Examples

```
data(dieselgate)
task <- EventStudyTask$new(dieselgate$firm, dieselgate$index, dieselgate$request)
task <- run_event_study(task, ParameterSet$new())
advice <- recommend_stat(task)
print(advice)
```

report_table	<i>Render a data frame as a styled table (HTML / PDF / Word / Markdown)</i>
--------------	---

Description

Internal rendering helper used by the event study report skeleton and `es_report()`. Exported so `inst/rmarkdown/` templates can call it without `:::`.

Usage

```
report_table(x, caption = NULL, col.names = NULL, digits = NULL)
```

Arguments

<code>x</code>	A data frame to render.
<code>caption</code>	Optional character scalar caption, or <code>NULL</code> .
<code>col.names</code>	Optional character vector of display column names, or <code>NULL</code> . When supplied, <code>x</code> is renamed before <code>tt()/kable()</code> so both rendering paths show the display names.
<code>digits</code>	Optional integer digit count, or <code>NULL</code> . Passed to <code>tt()</code> and <code>knitr::kable()</code> . Pass <code>NULL</code> for <code>pre-sprintf()</code> -formatted tables to preserve byte-compatible output.

Details

When **tinytable** is available (see `.tinytable_available()`), renders via `tinytable::tt()` with a bold header, auto-aligned columns, and optional digit rounding. Falls back to `knitr::kable()` when **tinytable** is absent – byte-identical to the prior scattered `knitr::kable()` call sites.

Value

`invisible(NULL)`. Side-effect: prints the table via `knit_print`.

ReturnCalculation	<i>ReturnCalculation</i>
-------------------	--------------------------

Description

Base (abstract) class for return calculation strategies. Subclass this to plug a custom return calculation into `prepare_event_study()` / `EventStudyTask$new()` via `ParameterSet$return_calculation`.

A subclass must implement `calculate_return(tbl, in_column, out_column)`: given a tibble of prices in `in_column`, add a column named `out_column` with the calculated return (e.g. simple or log return). See `SimpleReturn` and `LogReturn` for reference implementations.

Public fields

`name` Name of the return calculation.

Methods

Public methods:

- [ReturnCalculation\\$calculate_return\(\)](#)
- [ReturnCalculation\\$clone\(\)](#)

`ReturnCalculation$calculate_return()`: Calculates the return for a single stock.

Usage:

```
ReturnCalculation$calculate_return(
  tbl,
  in_column = "adjusted",
  out_column = "adjusted_return"
)
```

Arguments:

`tbl` The dataframe with the stock price.

`in_column` The column name of the price information.

`out_column` The column name where the return will be saved.

`ReturnCalculation$clone()`: The objects of this class are cloneable with this method.

Usage:

```
ReturnCalculation$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-models: [BHARModel](#), [Carhart4FactorModel](#), [ComparisonPeriodMeanAdjustedModel](#), [DCCGARCHModel](#), [FamaFrench3FactorModel](#), [FamaFrench5FactorModel](#), [GARCHModel](#), [LinearFactorModel](#), [LogReturn](#), [MarketAdjustedModel](#), [MarketModel](#), [ModelBase](#), [RollingWindowModel](#), [SimpleReturn](#), [VolatilityModel](#), [VolumeModel](#)

RollingWindowModel	<i>Rolling Window Model</i>
--------------------	-----------------------------

Description

Event study model with time-varying parameters estimated via a rolling OLS window over the estimation period. The last rolling window's parameters are used for event-window prediction. This captures parameter instability that is common in financial return data.

Super class

`ModelBase` -> `RollingWindowModel`

Public fields

`model_name` Name of the model.
`window_size` Rolling window size. Default 60.
`min_obs` Minimum observations required. Default 30.

Methods

Public methods:

- `RollingWindowModel$new()`
- `RollingWindowModel$fit()`
- `RollingWindowModel$abnormal_returns()`
- `RollingWindowModel$clone()`

`RollingWindowModel$new()`: Create a new `RollingWindowModel`.

Usage:

```
RollingWindowModel$new(window_size = 60L, min_obs = 30L)
```

Arguments:

`window_size` Size of the rolling window.
`min_obs` Minimum valid (complete-pair) observations required inside each individual rolling sub-window for that sub-window to be fit. Distinct from `validate_task()`'s `min_estimation_obs`, which is a whole-estimation-window advisory threshold checked once per event, not per sub-window.

`RollingWindowModel$fit()`: Fit the rolling window model on the estimation window.

Usage:

```
RollingWindowModel$fit(data_tbl)
```

Arguments:

`data_tbl` Data frame or tibble with `firm_returns`, `index_returns`, `estimation_window`, `event_window` columns.

`RollingWindowModel$abnormal_returns()`: Calculate abnormal returns using the last rolling window parameters.

Usage:

```
RollingWindowModel$abnormal_returns(data_tbl)
```

Arguments:

`data_tbl` Data frame or tibble.

`RollingWindowModel$clone()`: The objects of this class are cloneable with this method.

Usage:

```
RollingWindowModel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-models: [BHARModel](#), [Carhart4FactorModel](#), [ComparisonPeriodMeanAdjustedModel](#), [DCCGARCHModel](#), [FamaFrench3FactorModel](#), [FamaFrench5FactorModel](#), [GARCHModel](#), [LinearFactorModel](#), [LogReturn](#), [MarketAdjustedModel](#), [MarketModel](#), [ModelBase](#), [ReturnCalculation](#), [SimpleReturn](#), [VolatilityModel](#), [VolumeModel](#)

run_event_study	<i>Run a Complete Event Study</i>
-----------------	-----------------------------------

Description

Convenience wrapper that runs the full event study pipeline: prepare data, fit models, and calculate test statistics in a single call.

Usage

```
run_event_study(
  task,
  parameter_set = ParameterSet$new(),
  report = FALSE,
  report_args = list(),
  verbose = getOption("eventstudy.verbose", TRUE)
)
```

Arguments

<code>task</code>	An <code>EventStudyTask</code> object.
<code>parameter_set</code>	A <code>ParameterSet</code> object defining the event study. Defaults to a new <code>ParameterSet</code> with default settings.

report	Logical. When FALSE (default), the function behaves exactly as before this parameter was added. When TRUE, also renders a report via es_report() after the pipeline completes and attaches the rendered path(s) as attr(task, "report_path").
report_args	Named list forwarded to es_report() when report = TRUE. May include output_file, format, provider, and any other argument accepted by es_report() . Defaults to list() (uses es_report() defaults, which render "event_study_report.html" in the working directory).
verbose	Logical; if FALSE, suppress informational messages. Default <code>getOption("eventstudy.verbose", TRUE)</code> .

Value

The task object with all results computed. When report = TRUE, the returned task additionally carries attr(task, "report_path") with the path(s) of the rendered report file(s). The return type is always a fitted EventStudyTask in both paths.

See Also

[prepare_event_study](#), [fit_model](#), [calculate_statistics](#), [MarketModel](#), [ARTTest](#)

Other eventstudy-pipeline: [EventStudyTask](#), [ParameterSet](#), [calculate_statistics\(\)](#), [fit_model\(\)](#), [prepare_event_study\(\)](#)

Examples

```
data(dieselgate)
task <- EventStudyTask$new(dieselgate$firm, dieselgate$index, dieselgate$request)

# Run the pipeline and render a report to a temporary directory
out <- file.path(tempdir(), "event_study_report.html")
task <- run_event_study(task, ParameterSet$new(), report = TRUE,
  report_args = list(output_file = out))
attr(task, "report_path") # path to the rendered HTML
```

sc_placebo_test

Placebo Test for Synthetic Control

Description

Performs a placebo (permutation) test by re-estimating the synthetic control for each donor unit as a pseudo-treated unit. The p-value is the rank of the treated unit's RMSPE ratio among all units.

Usage

```
sc_placebo_test(task, n_placebo = NULL)
```

Arguments

`task` A `SyntheticControlTask` with results.
`n_placebo` Number of placebo units to use. Default NULL (all donors).

Value

The task with `results$placebo` populated, a list containing `rmspe_ratios`, `p_value`, and `placebo_gaps`.

See Also

Other eventstudy-tasks: [IntradayEventStudyTask](#), [PanelEventStudyTask](#), [SyntheticControlTask](#), [estimate_panel_event_study\(\)](#), [estimate_synthetic_control\(\)](#), [nonparametric_intraday_test\(\)](#), [plot_panel_event_study\(\)](#), [plot_synthetic_control\(\)](#), [prepare_intraday_event_study\(\)](#)

SignTest

Sign Test for Multiple Events

Description

Tests whether the proportion of positive abnormal returns differs significantly from 0.5 under the null hypothesis. The test statistic is approximately distributed as $N(0, 1)$.

Super class

[TestStatisticBase](#) -> SignTest

Public fields

`name` Short code of the test statistic.

Methods**Public methods:**

- [SignTest\\$compute\(\)](#)
- [SignTest\\$clone\(\)](#)

`SignTest$compute()`: Computes the sign test for multiple events.

Usage:

`SignTest$compute(data_tbl, model)`

Arguments:

`data_tbl` The data for a multiple event with calculated abnormal returns.

`model` The fitted model (unused).

`SignTest$clone()`: The objects of this class are cloneable with this method.

Usage:

`SignTest$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-statistics: [ARTTest](#), [BHARTTest](#), [BMPTTest](#), [CARTTest](#), [CSectTTest](#), [CalendarTimePortfolioTest](#), [GeneralizedSignTest](#), [KolariPynnonenTest](#), [MultiEventStatisticsSet](#), [PatellZTest](#), [RankTest](#), [SingleEventStatisticsSet](#), [StatisticsSetBase](#), [TestStatisticBase](#), [adjust_p_values\(\)](#), [bootstrap_test\(\)](#), [car_by_group\(\)](#), [car_quantiles\(\)](#), [cross_sectional_regression\(\)](#), [model_diagnostics\(\)](#), [pretrend_test\(\)](#), [simulate_event_study\(\)](#), [validate_task\(\)](#)

SimpleReturn

*R6 class for simple return calculation***Description**

R6 class for simple return calculation

Super class

[ReturnCalculation](#) -> SimpleReturn

Public fields

name Name of the return calculation.

Methods**Public methods:**

- [SimpleReturn\\$calculate_return\(\)](#)
- [SimpleReturn\\$clone\(\)](#)

[SimpleReturn\\$calculate_return\(\)](#): Calculates the simple return for a single stock.

Usage:

```
SimpleReturn$calculate_return(
  tbl,
  in_column = "adjusted",
  out_column = "adjusted_return"
)
```

Arguments:

tbl The dataframe with the stock price.

in_column The column name of the price information.

out_column The column name where the return will be saved.

[SimpleReturn\\$clone\(\)](#): The objects of this class are cloneable with this method.

Usage:

```
SimpleReturn$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

See Also

Other eventstudy-models: [BHARModel](#), [Carhart4FactorModel](#), [ComparisonPeriodMeanAdjustedModel](#), [DCCGARCHModel](#), [FamaFrench3FactorModel](#), [FamaFrench5FactorModel](#), [GARCHModel](#), [LinearFactorModel](#), [LogReturn](#), [MarketAdjustedModel](#), [MarketModel](#), [ModelBase](#), [ReturnCalculation](#), [RollingWindowModel](#), [VolatilityModel](#), [VolumeModel](#)

simulate_event_study *Simulate Event Study for Power Analysis*

Description

Runs Monte Carlo simulations to estimate the statistical power (rejection rate) of an event study design. Each simulation generates synthetic data from a user-specified DGP, runs the full event study pipeline, and records whether the null hypothesis is rejected.

Usage

```
simulate_event_study(
  n_events = 20,
  event_window = c(-5, 5),
  estimation_window_length = 120,
  abnormal_return = 0,
  return_model = MarketModel$new(),
  test_statistic = "CSectT",
  alpha = 0.05,
  n_simulations = 1000,
  dgp_params = list(),
  seed = NULL
)
```

Arguments

n_events	Number of events (firms) per simulation. Default 20.
event_window	Event window as c(start, end). Default c(-5, 5).
estimation_window_length	Length of the estimation window. Default 120.
abnormal_return	Injected abnormal return on the event day (day 0). Set to 0 for size analysis, positive for power analysis. Default 0.
return_model	An initialized return model object. Default MarketModel\$new().
test_statistic	Name of the multi-event test statistic to evaluate. Default "CSectT".
alpha	Significance level. Default 0.05.
n_simulations	Number of Monte Carlo replications. Default 1000.
dgp_params	List of DGP parameters: alpha (drift), beta (market exposure), sigma_firm (idiosyncratic volatility), sigma_market (market volatility).
seed	Optional seed for reproducibility.

Value

An S3 object of class `es_simulation` with components:

power Rejection rate at the event day

rejection_by_day Tibble of rejection rates for each event-window day

test_stats Vector of test statistics at the event day from each sim

params List of simulation parameters

See Also

Other eventstudy-statistics: [ARTTest](#), [BHARTTest](#), [BMPTTest](#), [CARTTest](#), [CSectTTest](#), [CalendarTimePortfolioTest](#), [GeneralizedSignTest](#), [KolariPynnonenTest](#), [MultiEventStatisticsSet](#), [PatellZTest](#), [RankTest](#), [SignTest](#), [SingleEventStatisticsSet](#), [StatisticsSetBase](#), [TestStatisticBase](#), [adjust_p_values\(\)](#), [bootstrap_test\(\)](#), [car_by_group\(\)](#), [car_quantiles\(\)](#), [cross_sectional_regression\(\)](#), [model_diagnostics\(\)](#), [pretrend_test\(\)](#), [validate_task\(\)](#)

SingleEventStatisticsSet

Single Event Statistic Set

Description

Contains a set of single event test statistics as, e.g., AR and CAR t test. These are the default test statistics.

Super class

[StatisticsSetBase](#) -> SingleEventStatisticsSet

Public fields

tests Container with the statistical tests.

Methods**Public methods:**

- [SingleEventStatisticsSet\\$clone\(\)](#)

`SingleEventStatisticsSet$clone()`: The objects of this class are cloneable with this method.

Usage:

`SingleEventStatisticsSet$clone(deep = FALSE)`

Arguments:

deep Whether to make a deep clone.

See Also

Other eventstudy-statistics: [ARTTest](#), [BHARTTest](#), [BMPTTest](#), [CARTTest](#), [CSectTTest](#), [CalendarTimePortfolioTest](#), [GeneralizedSignTest](#), [KolariPynnonenTest](#), [MultiEventStatisticsSet](#), [PatellZTest](#), [RankTest](#), [SignTest](#), [StatisticsSetBase](#), [TestStatisticBase](#), [adjust_p_values\(\)](#), [bootstrap_test\(\)](#), [car_by_group\(\)](#), [car_quantiles\(\)](#), [cross_sectional_regression\(\)](#), [model_diagnostics\(\)](#), [pretrend_test\(\)](#), [simulate_event_study\(\)](#), [validate_task\(\)](#)

StatisticsSetBase	<i>Adds a test statistic to this container.</i>
-------------------	---

Description

Base class for test statistic container

Public fields

`tests` Container with the statistical tests. Test container initialization

Methods**Public methods:**

- [StatisticsSetBase\\$new\(\)](#)
- [StatisticsSetBase\\$add_test\(\)](#)
- [StatisticsSetBase\\$clone\(\)](#)

`StatisticsSetBase$new()`: Initializes a test statistic container. Single event and multiple event test statistics are collected in separated containers.

Usage:

```
StatisticsSetBase$new(tests = NULL)
```

Arguments:

`tests` List of test statistics.

`StatisticsSetBase$add_test()`:

Usage:

```
StatisticsSetBase$add_test(test)
```

Arguments:

`test` A single event study tests.

`StatisticsSetBase$clone()`: The objects of this class are cloneable with this method.

Usage:

```
StatisticsSetBase$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-statistics: [ARTTest](#), [BHARTTest](#), [BMPTTest](#), [CARTTest](#), [CSectTTest](#), [CalendarTimePortfolioTest](#), [GeneralizedSignTest](#), [KolariPynnonenTest](#), [MultiEventStatisticsSet](#), [PatellZTest](#), [RankTest](#), [SignTest](#), [SingleEventStatisticsSet](#), [TestStatisticBase](#), [adjust_p_values\(\)](#), [bootstrap_test\(\)](#), [car_by_group\(\)](#), [car_quantiles\(\)](#), [cross_sectional_regression\(\)](#), [model_diagnostics\(\)](#), [pretrend_test\(\)](#), [simulate_event_study\(\)](#), [validate_task\(\)](#)

SyntheticControlTask *Synthetic Control Task*

Description

Task container for synthetic control analysis. Holds treated unit data, donor pool data, treatment time, and results after estimation.

Public fields

`treated_data` Tibble with time and outcome columns for the treated unit.
`donor_data` Tibble in long format with unit, time, and outcome columns for donor units.
`treatment_time` The time period when treatment begins.
`results` Estimation results (populated after estimation).

Methods**Public methods:**

- [SyntheticControlTask\\$new\(\)](#)
- [SyntheticControlTask\\$print\(\)](#)
- [SyntheticControlTask\\$clone\(\)](#)

`SyntheticControlTask$new()`: Create a new `SyntheticControlTask`.

Usage:

```
SyntheticControlTask$new(treated_data, donor_data, treatment_time)
```

Arguments:

`treated_data` Tibble with time and outcome.
`donor_data` Long-format tibble with unit, time, outcome.
`treatment_time` Period when treatment begins.

`SyntheticControlTask$print()`: Print summary.

Usage:

```
SyntheticControlTask$print(...)
```

`SyntheticControlTask$clone()`: The objects of this class are cloneable with this method.

Usage:

```
SyntheticControlTask$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-tasks: [IntradayEventStudyTask](#), [PanelEventStudyTask](#), [estimate_panel_event_study\(\)](#), [estimate_synthetic_control\(\)](#), [nonparametric_intraday_test\(\)](#), [plot_panel_event_study\(\)](#), [plot_synthetic_control\(\)](#), [prepare_intraday_event_study\(\)](#), [sc_placebo_test\(\)](#)

TestStatisticBase	<i>TestStatisticBase</i>
-------------------	--------------------------

Description

Base (abstract) class for single-event and multi-event test statistics. Subclass this to plug a custom test statistic into `calculate_statistics()` / `run_event_study()`.

A subclass must implement `compute(data_tbl, model)`:

- `data_tbl` – a single event’s rows with an `abnormal_returns` column already populated by the fitted model.
- `model` – the fitted `ModelBase` (or `list-mock`) instance for that event; its `statistics` field carries `sigma`, `degree_of_freedom`, `residuals` and (since 2026-09-24) `n_params`.

`compute()` returns a tibble with the statistic’s result columns for that event; multi-event subclasses (see `R/multi_event_test_statistics.R`) aggregate across all events in a group.

`confidence_level` and `confidence_type` are validated and stored at construction (`confidence_type` must be one of “two-sided” (default), “less” or “greater”); every p-value computed anywhere in `EventStudy` is currently two-sided, so a non-default `confidence_type` is accepted but ignored, with one warning at construction (A10, 2026-09-24).

Public fields

`name` Short code of the test statistic.

`confidence_level` The chosen confidence level.

`confidence_type` Side of the test. One of “two-sided” (default), “less” or “greater”. A10 (2026-09-24): every p-value computed anywhere in `EventStudy` is currently two-sided; a non-default value is validated but has NO effect on any computed statistic – it is stored and a one-time warning is emitted at construction. One-sided p-values are not implemented.

Methods**Public methods:**

- [TestStatisticBase\\$new\(\)](#)
- [TestStatisticBase\\$compute\(\)](#)
- [TestStatisticBase\\$clone\(\)](#)

`TestStatisticBase$new()`: Initializes the test statistic. This includes the confidence level and the type of the test (‘less’, ‘greater’ or ‘two-sided’)

Usage:

```
TestStatisticBase$new(confidence_level = 0.95, confidence_type = "two-sided")
```

Arguments:

`confidence_level` The confidence level for the confidence band. Must be a number between 0 and 1.

`confidence_type` Side of the test statistic: "two-sided" (default), "less" or "greater". Every p-value in EventStudy is currently two-sided; supplying "less" or "greater" emits one warning that the value is stored but ignored by every `compute()` method.

`TestStatisticBase$compute()`: Computes the test statistics for a single event.

Usage:

```
TestStatisticBase$compute(data_tbl, model)
```

Arguments:

`data_tbl` The data for a single event with calculated abnormal returns.

`model` The fitted model that includes the necessary information for calculating the test statistic.

`TestStatisticBase$clone()`: The objects of this class are cloneable with this method.

Usage:

```
TestStatisticBase$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-statistics: [ARTTest](#), [BHARTTest](#), [BMPTTest](#), [CARTTest](#), [CSectTTest](#), [CalendarTimePortfolioTest](#), [GeneralizedSignTest](#), [KolariPynnonenTest](#), [MultiEventStatisticsSet](#), [PatellZTest](#), [RankTest](#), [SignTest](#), [SingleEventStatisticsSet](#), [StatisticsSetBase](#), [adjust_p_values\(\)](#), [bootstrap_test\(\)](#), [car_by_group\(\)](#), [car_quantiles\(\)](#), [cross_sectional_regression\(\)](#), [model_diagnostics\(\)](#), [pretrend_test\(\)](#), [simulate_event_study\(\)](#), [validate_task\(\)](#)

theme_eventstudy	<i>EventStudy ggplot2 Theme</i>
------------------	---------------------------------

Description

A clean, publication-ready ggplot2 theme for EventStudy plots. Built on [theme_minimal](#) with a centred plot title, a bottom legend, and subtle gridlines styled to the EventStudy palette.

Usage

```
theme_eventstudy(base_size = 11, base_family = "")
```

Arguments

`base_size` Base font size in points. Default is 11.

`base_family` Base font family. Default "" uses the R device default, which works on all platforms without requiring installed fonts. Pass a family name (e.g. "Inter") if that font is installed on the target device.

Value

A [theme](#) object that composes with `+`.

See Also

[es_colours](#), [plot_event_study](#)

Other eventstudy-plots: [es_colours](#), [plot_car_distribution\(\)](#), [plot_diagnostics\(\)](#), [plot_event_study\(\)](#), [plot_stocks\(\)](#)

tidy.EventStudyTask	<i>Tidy Event Study Results</i>
---------------------	---------------------------------

Description

Extract event study results in a tidy (long-format) tibble compatible with broom conventions. This enables integration with standard tidyverse workflows.

Usage

```
tidy.EventStudyTask(
  x,
  type = c("ar", "car", "aar", "model"),
  stat_name = "CSectT",
  ...
)
```

Arguments

<code>x</code>	An EventStudyTask object.
<code>type</code>	Type of results to extract: "ar" (abnormal returns), "car" (cumulative abnormal returns), "aar" (average abnormal returns with test statistics), or "model" (model fit statistics).
<code>stat_name</code>	For type "aar", the name of the multi-event test statistic. Defaults to "CSectT".
<code>...</code>	Additional arguments (unused).

Value

A tibble with columns following broom conventions:

term Identifier for the observation (e.g., `relative_index`)

estimate The estimated value (AR, CAR, AAR, or coefficient)

std.error Standard error where available

statistic Test statistic value

p.value p-value where available

See Also

Other eventstudy-export: [es_report\(\)](#), [export_results\(\)](#), [generate_report\(\)](#)

validate_task	<i>Validate an Event Study experiment</i>
---------------	---

Description

Performs validation checks on the event study task: 1. Does the event date exist in the stock data for each event? 2. Are there sufficient observations in the estimation window? 3. Are there any gaps in the time series? 4. Do estimation and event windows overlap?

Usage

```
validate_task(
  task,
  parameter_set = NULL,
  min_estimation_obs = 30,
  verbose = getOption("eventstudy.verbose", TRUE)
)
```

Arguments

task	The event study task.
parameter_set	The parameter set that defines the event study.
min_estimation_obs	Minimum number of observations required in the <i>whole</i> per-event estimation window (a task-level validation threshold checked once per event). Default is 30. Distinct from <code>RollingWindowModel\$min_obs</code> , which bounds each individual sub-window inside a single event's rolling-window fit.
verbose	Logical; if FALSE, suppress informational messages. Default <code>getOption("eventstudy.verbose", TRUE)</code> . Warnings are never gated.

Value

The task object (invisibly). Warnings are issued for each problem found.

See Also

Other eventstudy-statistics: [ARTTest](#), [BHARTTest](#), [BMPTTest](#), [CARTTest](#), [CSectTTest](#), [CalendarTimePortfolioTest](#), [GeneralizedSignTest](#), [KolariPynnonenTest](#), [MultiEventStatisticsSet](#), [PatellZTest](#), [RankTest](#), [SignTest](#), [SingleEventStatisticsSet](#), [StatisticsSetBase](#), [TestStatisticBase](#), [adjust_p_values\(\)](#), [bootstrap_test\(\)](#), [car_by_group\(\)](#), [car_quantiles\(\)](#), [cross_sectional_regression\(\)](#), [model_diagnostics\(\)](#), [pretrend_test\(\)](#), [simulate_event_study\(\)](#)

VolatilityModel

Volatility Event Study Model

Description

Model for volatility-based event studies. Computes abnormal volatility as the ratio of event-window squared returns to estimation-window variance. The abnormal measure is written to the `abnormal_returns` column for compatibility with existing test statistics.

Super class

`ModelBase` -> `VolatilityModel`

Public fields

`model_name` Name of the model.

Methods

Public methods:

- `VolatilityModel$fit()`
- `VolatilityModel$abnormal_returns()`
- `VolatilityModel$clone()`

`VolatilityModel$fit()`: Fit the volatility model. Estimates expected variance from estimation window.

Usage:

`VolatilityModel$fit(data_tbl)`

Arguments:

`data_tbl` Data frame or tibble.

`VolatilityModel$abnormal_returns()`: Calculate abnormal volatility (squared returns / expected variance - 1).

Usage:

`VolatilityModel$abnormal_returns(data_tbl)`

Arguments:

`data_tbl` Data frame or tibble.

`VolatilityModel$clone()`: The objects of this class are cloneable with this method.

Usage:

`VolatilityModel$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-models: [BHARModel](#), [Carhart4FactorModel](#), [ComparisonPeriodMeanAdjustedModel](#), [DCCGARCHModel](#), [FamaFrench3FactorModel](#), [FamaFrench5FactorModel](#), [GARCHModel](#), [LinearFactorModel](#), [LogReturn](#), [MarketAdjustedModel](#), [MarketModel](#), [ModelBase](#), [ReturnCalculation](#), [RollingWindowModel](#), [SimpleReturn](#), [VolumeModel](#)

VolumeModel

Volume Event Study Model

Description

Model for volume-based event studies. Computes abnormal volume as the difference between observed volume and expected volume from the estimation window mean. The data must contain a `firm_volume` column (and optionally `index_volume` for market-adjusted volume).

The existing test statistics infrastructure works on the `abnormal_returns` column, so this model writes abnormal volume to that same column for compatibility.

Super class

[ModelBase](#) -> VolumeModel

Public fields

`model_name` Name of the model.

`log_transform` Whether to log-transform volume. Default TRUE.

Methods**Public methods:**

- [VolumeModel\\$new\(\)](#)
- [VolumeModel\\$fit\(\)](#)
- [VolumeModel\\$abnormal_returns\(\)](#)
- [VolumeModel\\$clone\(\)](#)

`VolumeModel$new()`: Create a new VolumeModel.

Usage:

```
VolumeModel$new(log_transform = TRUE)
```

Arguments:

`log_transform` Whether to log-transform volume before analysis.

`VolumeModel$fit()`: Fit the volume model. Computes expected volume from estimation window.

Usage:

```
VolumeModel$fit(data_tbl)
```

Arguments:

`data_tbl` Data frame or tibble with `firm_volume` column.

`VolumeModel$abnormal_returns()`: Calculate abnormal volume.

Usage:

```
VolumeModel$abnormal_returns(data_tbl)
```

Arguments:

`data_tbl` Data frame or tibble.

`VolumeModel$clone()`: The objects of this class are cloneable with this method.

Usage:

```
VolumeModel$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

Other eventstudy-models: [BHARModel](#), [Carhart4FactorModel](#), [ComparisonPeriodMeanAdjustedModel](#), [DCCGARCHModel](#), [FamaFrench3FactorModel](#), [FamaFrench5FactorModel](#), [GARCHModel](#), [LinearFactorModel](#), [LogReturn](#), [MarketAdjustedModel](#), [MarketModel](#), [ModelBase](#), [ReturnCalculation](#), [RollingWindowModel](#), [SimpleReturn](#), [VolatilityModel](#)

Index

- * **datasets**
 - dieselgate, [25](#)
 - earnings_surprises, [28](#)
- * **eventstudy-advisor**
 - AnthropicProvider, [5](#)
 - CustomProvider, [21](#)
 - es_advise, [32](#)
 - es_diagnostics, [34](#)
 - es_kb, [35](#)
 - flag_robustness, [46](#)
 - OpenAICompatProvider, [67](#)
 - provider, [82](#)
 - ProviderBase, [83](#)
 - recommend_stat, [86](#)
- * **eventstudy-export**
 - es_report, [36](#)
 - export_results, [42](#)
 - generate_report, [52](#)
 - tidy.EventStudyTask, [100](#)
- * **eventstudy-models**
 - BHARModel, [8](#)
 - Carhart4FactorModel, [14](#)
 - ComparisonPeriodMeanAdjustedModel, [18](#)
 - DCCGARCHModel, [22](#)
 - FamaFrench3FactorModel, [43](#)
 - FamaFrench5FactorModel, [44](#)
 - GARCHModel, [50](#)
 - LinearFactorModel, [57](#)
 - LogReturn, [58](#)
 - MarketAdjustedModel, [59](#)
 - MarketModel, [61](#)
 - ModelBase, [62](#)
 - ReturnCalculation, [88](#)
 - RollingWindowModel, [89](#)
 - SimpleReturn, [93](#)
 - VolatilityModel, [102](#)
 - VolumeModel, [103](#)
- * **eventstudy-pipeline**
 - calculate_statistics, [12](#)
 - EventStudyTask, [39](#)
 - fit_model, [45](#)
 - ParameterSet, [71](#)
 - prepare_event_study, [78](#)
 - run_event_study, [90](#)
- * **eventstudy-plots**
 - es_colours, [34](#)
 - plot_car_distribution, [73](#)
 - plot_diagnostics, [74](#)
 - plot_event_study, [74](#)
 - plot_stocks, [76](#)
 - theme_eventstudy, [99](#)
- * **eventstudy-statistics**
 - adjust_p_values, [4](#)
 - ARTTest, [7](#)
 - BHARTTest, [9](#)
 - BMPTTest, [10](#)
 - bootstrap_test, [11](#)
 - CalendarTimePortfolioTest, [13](#)
 - car_by_group, [16](#)
 - car_quantiles, [17](#)
 - CARTTest, [15](#)
 - cross_sectional_regression, [19](#)
 - CSectTTest, [20](#)
 - GeneralizedSignTest, [51](#)
 - KolariPynnonenTest, [56](#)
 - model_diagnostics, [64](#)
 - MultiEventStatisticsSet, [65](#)
 - PatellZTest, [72](#)
 - pretrend_test, [79](#)
 - RankTest, [85](#)
 - SignTest, [92](#)
 - simulate_event_study, [94](#)
 - SingleEventStatisticsSet, [95](#)
 - StatisticsSetBase, [96](#)
 - TestStatisticBase, [98](#)
 - validate_task, [101](#)
- * **eventstudy-tasks**

- estimate_panel_event_study, 30
- estimate_synthetic_control, 31
- IntradayEventStudyTask, 54
- nonparametric_intraday_test, 66
- PanelEventStudyTask, 69
- plot_panel_event_study, 75
- plot_synthetic_control, 77
- prepare_intraday_event_study, 78
- sc_placebo_test, 91
- SyntheticControlTask, 97
- .Deprecated, 38

- adjust_p_values, 4
- adjust_p_values(), 8, 10–12, 14, 16, 17, 20, 21, 52, 57, 64, 65, 73, 79, 85, 93, 95–97, 99, 101
- AnthropicProvider, 5, 22, 33, 35, 36, 47, 69, 83, 84, 87
- ARTTest, 5, 7, 10–12, 14, 16, 17, 20, 21, 52, 57, 64, 65, 73, 79, 85, 91, 93, 95–97, 99, 101

- BHARModel, 8, 15, 19, 23, 44, 45, 51, 58–60, 62, 64, 88, 90, 94, 103, 104
- BHARTTest, 5, 8, 9, 11, 12, 14, 16, 17, 20, 21, 52, 57, 64, 65, 73, 79, 85, 93, 95–97, 99, 101
- BMPTest, 5, 8, 10, 10, 12, 14, 16, 17, 20, 21, 52, 56, 57, 64, 65, 73, 79, 85, 93, 95–97, 99, 101
- bootstrap_test, 11
- bootstrap_test(), 5, 8, 10, 11, 14, 16, 17, 20, 21, 52, 57, 64, 65, 73, 79, 85, 93, 95–97, 99, 101

- calculate_statistics, 12, 46, 78, 91
- calculate_statistics(), 42, 46, 72, 78, 91
- CalendarTimePortfolioTest, 5, 8, 10–12, 13, 16, 17, 20, 21, 52, 57, 64, 65, 73, 79, 85, 93, 95–97, 99, 101
- car_by_group, 16
- car_by_group(), 5, 8, 10–12, 14, 16, 17, 20, 21, 52, 57, 64, 65, 73, 79, 85, 93, 95–97, 99, 101
- car_quantiles, 17
- car_quantiles(), 5, 8, 10–12, 14, 16, 17, 20, 21, 52, 57, 64, 65, 73, 79, 85, 93, 95–97, 99, 101

- Carhart4FactorModel, 9, 14, 19, 23, 44, 45, 51, 58–60, 62, 64, 88, 90, 94, 103, 104
- CARTTest, 5, 8, 10–12, 14, 15, 17, 20, 21, 52, 57, 64, 65, 73, 79, 85, 93, 95–97, 99, 101
- ComparisonPeriodMeanAdjustedModel, 9, 15, 18, 23, 44, 45, 51, 58–60, 62, 64, 88, 90, 94, 103, 104
- cross_sectional_regression, 19
- cross_sectional_regression(), 5, 8, 10–12, 14, 16, 17, 21, 52, 57, 64, 65, 73, 79, 85, 93, 95–97, 99, 101
- CSectTTest, 5, 8, 10–12, 14, 16, 17, 20, 20, 52, 57, 64, 65, 73, 79, 85, 93, 95–97, 99, 101
- CustomProvider, 7, 21, 33, 35, 36, 47, 69, 83, 84, 87

- DCCGARCHModel, 9, 15, 19, 22, 44, 45, 51, 58–60, 62, 64, 88, 90, 94, 103, 104
- degenerate-input-contract, 24
- dieselgate, 25
- download_factor_data, 26
- download_risk_free_rate, 27
- download_stock_data, 26, 28, 29

- earnings_surprises, 28
- es_advise, 32, 53
- es_advise(), 7, 22, 35, 36, 47, 69, 83, 84, 87
- es_colours, 34, 74, 75, 77, 100
- es_diagnostics, 32, 33, 34, 35, 36, 38, 47, 87
- es_diagnostics(), 7, 22, 33, 36, 47, 69, 83, 84, 87
- es_kb, 35, 47, 87
- es_kb(), 7, 22, 33, 35, 47, 69, 83, 84, 87
- es_report, 36, 91
- es_report(), 43, 54, 101
- estimate_panel_event_study, 30
- estimate_panel_event_study(), 32, 55, 67, 70, 76, 77, 79, 92, 98
- estimate_synthetic_control, 31
- estimate_synthetic_control(), 31, 55, 67, 70, 76, 77, 79, 92, 98
- eventstudy-deprecation, 38
- eventstudy-shape-contracts, 39
- EventStudyTask, 12, 25, 29, 39, 46, 54, 55, 72, 78, 91
- export_results, 42

- `export_results()`, 38, 54, 101
- `FamaFrench3FactorModel`, 9, 15, 19, 23, 43, 45, 51, 58–60, 62, 64, 88, 90, 94, 103, 104
- `FamaFrench5FactorModel`, 9, 15, 19, 23, 44, 45, 51, 58–60, 62, 64, 88, 90, 94, 103, 104
- `fit_model`, 12, 45, 78, 91
- `fit_model()`, 12, 42, 72, 78, 91
- `flag_robustness`, 33, 35, 36, 46, 87
- `flag_robustness()`, 7, 22, 33, 35, 36, 69, 83, 84, 87
- `format.Advice`, 47
- `format.es_advice`, 48
- `format.es_cross_sectional`, 48
- `format.es_diagnostics`, 49
- `format.es_simulation`, 49
- `format.EventStudySummary`, 50
- `GARCHModel`, 9, 15, 19, 23, 44, 45, 50, 58–60, 62, 64, 88, 90, 94, 103, 104
- `GeneralizedSignTest`, 5, 8, 10–12, 14, 16, 17, 20, 21, 51, 57, 64, 65, 73, 79, 85, 93, 95–97, 99, 101
- `generate_report`, 36, 38, 52
- `generate_report()`, 38, 43, 101
- `IntradayEventStudyTask`, 31, 32, 54, 67, 70, 76, 77, 79, 92, 98
- `KolariPynnonenTest`, 5, 8, 10–12, 14, 16, 17, 20, 21, 52, 56, 64, 65, 73, 79, 85, 93, 95–97, 99, 101
- `LinearFactorModel`, 9, 14, 15, 19, 23, 43–45, 51, 57, 59, 60, 62, 64, 88, 90, 94, 103, 104
- `LogReturn`, 9, 15, 19, 23, 44, 45, 51, 58, 58, 60, 62, 64, 88, 90, 94, 103, 104
- `MarketAdjustedModel`, 9, 15, 19, 23, 44, 45, 51, 58, 59, 59, 62, 64, 88, 90, 94, 103, 104
- `MarketModel`, 9, 15, 19, 23, 25, 44, 45, 51, 58–60, 61, 64, 88, 90, 91, 94, 103, 104
- `model_diagnostics`, 35, 64
- `model_diagnostics()`, 5, 8, 10–12, 14, 16, 17, 20, 21, 52, 57, 65, 73, 79, 85, 93, 95–97, 99, 101
- `ModelBase`, 8, 9, 14, 15, 18, 19, 22, 23, 43–45, 50, 51, 57–62, 62, 88–90, 94, 102–104
- `MultiEventStatisticsSet`, 5, 8, 10–12, 14, 16, 17, 20, 21, 52, 57, 64, 65, 73, 79, 85, 93, 95–97, 99, 101
- `nonparametric_intraday_test`, 66
- `nonparametric_intraday_test()`, 31, 32, 55, 70, 76, 77, 79, 92, 98
- `OpenAICompatProvider`, 7, 22, 33, 35, 36, 47, 67, 83, 84, 87
- `p.adjust`, 4, 5
- `PanelEventStudyTask`, 31, 32, 55, 67, 69, 76, 77, 79, 92, 98
- `ParameterSet`, 12, 25, 39, 42, 46, 71, 78, 91
- `PatellIZTest`, 5, 8, 10–12, 14, 16, 17, 20, 21, 52, 57, 64, 65, 72, 79, 85, 93, 95–97, 99, 101
- `plot_car_distribution`, 73
- `plot_car_distribution()`, 34, 74, 75, 77, 100
- `plot_diagnostics`, 74
- `plot_diagnostics()`, 34, 74, 75, 77, 100
- `plot_event_study`, 34, 74, 100
- `plot_event_study()`, 34, 74, 77, 100
- `plot_panel_event_study`, 75
- `plot_panel_event_study()`, 31, 32, 55, 67, 70, 77, 79, 92, 98
- `plot_stocks`, 38, 76
- `plot_stocks()`, 34, 74, 75, 100
- `plot_synthetic_control`, 77
- `plot_synthetic_control()`, 31, 32, 55, 67, 70, 76, 79, 92, 98
- `prepare_event_study`, 12, 46, 78, 91
- `prepare_event_study()`, 12, 42, 46, 72, 91
- `prepare_intraday_event_study`, 78
- `prepare_intraday_event_study()`, 31, 32, 55, 67, 70, 76, 77, 92, 98
- `pretrend_test`, 79
- `pretrend_test()`, 5, 8, 10–12, 14, 16, 17, 20, 21, 52, 57, 64, 65, 73, 85, 93, 95–97, 99, 101
- `print.Advice`, 80

- `print.es_advice`, 80
- `print.es_cross_sectional`, 81
- `print.es_diagnostics`, 49, 81
- `print.EventStudySummary`, 50, 82
- `provider`, 32, 33, 82
- `provider()`, 7, 22, 33, 35, 36, 47, 69, 84, 87
- `ProviderBase`, 5, 7, 21, 22, 33, 35, 36, 47, 68, 69, 83, 83, 87
- `RankTest`, 5, 8, 10–12, 14, 16, 17, 20, 21, 52, 57, 64, 65, 73, 79, 85, 93, 95–97, 99, 101
- `recommend_stat`, 33, 35, 36, 47, 86
- `recommend_stat()`, 7, 22, 33, 35, 36, 47, 69, 83, 84
- `report_table`, 87
- `ReturnCalculation`, 9, 15, 19, 23, 44, 45, 51, 58–60, 62, 64, 88, 90, 93, 94, 103, 104
- `RollingWindowModel`, 9, 15, 19, 23, 44, 45, 51, 58–60, 62, 64, 88, 89, 94, 103, 104
- `run_event_study`, 12, 33, 35, 38, 43, 46, 78, 90
- `run_event_study()`, 12, 42, 46, 72, 78
- `sc_placebo_test`, 91
- `sc_placebo_test()`, 31, 32, 55, 67, 70, 76, 77, 79, 98
- `SignTest`, 5, 8, 10–12, 14, 16, 17, 20, 21, 52, 57, 64, 65, 73, 79, 85, 92, 95–97, 99, 101
- `SimpleReturn`, 9, 15, 19, 23, 44, 45, 51, 58–60, 62, 64, 88, 90, 93, 103, 104
- `simulate_event_study`, 94
- `simulate_event_study()`, 5, 8, 10–12, 14, 16, 17, 20, 21, 52, 57, 64, 65, 73, 79, 85, 93, 96, 97, 99, 101
- `SingleEventStatisticsSet`, 5, 8, 10–12, 14, 16, 17, 20, 21, 52, 57, 64, 65, 73, 79, 85, 93, 95, 95, 97, 99, 101
- `StatisticsSetBase`, 5, 8, 10–12, 14, 16, 17, 20, 21, 52, 57, 64, 65, 73, 79, 85, 93, 95, 96, 96, 99, 101
- `SyntheticControlTask`, 31, 32, 55, 67, 70, 76, 77, 79, 92, 97
- `TestStatisticBase`, 5, 7–17, 20, 21, 51, 52, 56, 57, 64, 65, 72, 73, 79, 85, 92, 93, 95–97, 98, 101
- `theme`, 100
- `theme_eventstudy`, 34, 99
- `theme_eventstudy()`, 34, 74, 75, 77
- `theme_minimal`, 99
- `tidy.EventStudyTask`, 43, 100
- `tidy.EventStudyTask()`, 38, 43, 54
- `validate_task`, 101
- `validate_task()`, 5, 8, 10–12, 14, 16, 17, 20, 21, 52, 57, 64, 65, 73, 79, 85, 93, 95–97, 99
- `VolatilityModel`, 9, 15, 19, 23, 44, 45, 51, 58–60, 62, 64, 88, 90, 94, 102, 104
- `VolumeModel`, 9, 15, 19, 23, 44, 45, 51, 58–60, 62, 64, 88, 90, 94, 103, 103