Event Study

Release 0.1a

Jean-Baptiste Lemaire

Dec 14, 2019

CONTENTS

Event Study package is an open-source python project created to facilitate the computation of financial event study analysis.

CHAPTER

ONE

INSTALL

```
$ pip install eventstudy
```

CHAPTER

TWO

LEARN HOW TO USE

Go through the *Get started section* to discover through simple examples how to use the `eventstudy` package to run your event study for a single event or a sample of events.

Read the *API* for more details on functions and their parameters.

CHAPTER

THREE

PLAY WITH THE INTERACTIVE INTERFACE

A user-friendly interface has been developped using streamlit and can be accessed here.

CHAPTER

FOUR

USER GUIDE

4.1 Get Started

Through two examples, we will discover how to perform an event study analysis on a single event or on a sample of event.

Note: You can use the interactive version of this tutorial to play yourself with the functions.

4.1.1 Preliminary work

1. Load the `eventstudy` module and its dependencies: `numpy` and `matplotlib`:

```
import eventstudy as es
import numpy as np
import matplotlib.pyplot as plt
```

2. Set the parameters needed for your events: the returns and Fama-French factors (using `es.EventStudy.import_returns()` and `es.EventStudy.import_FamaFrench()`):

```
es.EventStudy.import_returns('returns.csv')
es.EventStudy.import_FamaFrench('famafrench.csv')
```

4.1.2 Example 1: A single event

As an introductory example, we will compute the event study analysis of the announcement of the first iphone, made by Steve Jobs during MacWorld exhibition, on January 7, 2007.

1. Run the event study, here using the Fama-French 3-factor model:

```
event = es.EventStudy.FamaFrench_3factor(
    security_ticker = 'AAPL',
    event_date = np.datetime64('2013-03-04'),
    event_window = (-2,+10),
    estimation_size = 300,
    buffer_size = 30
)
```

Note: You can easily play with the parameter and adjust the event study analysis to your need.

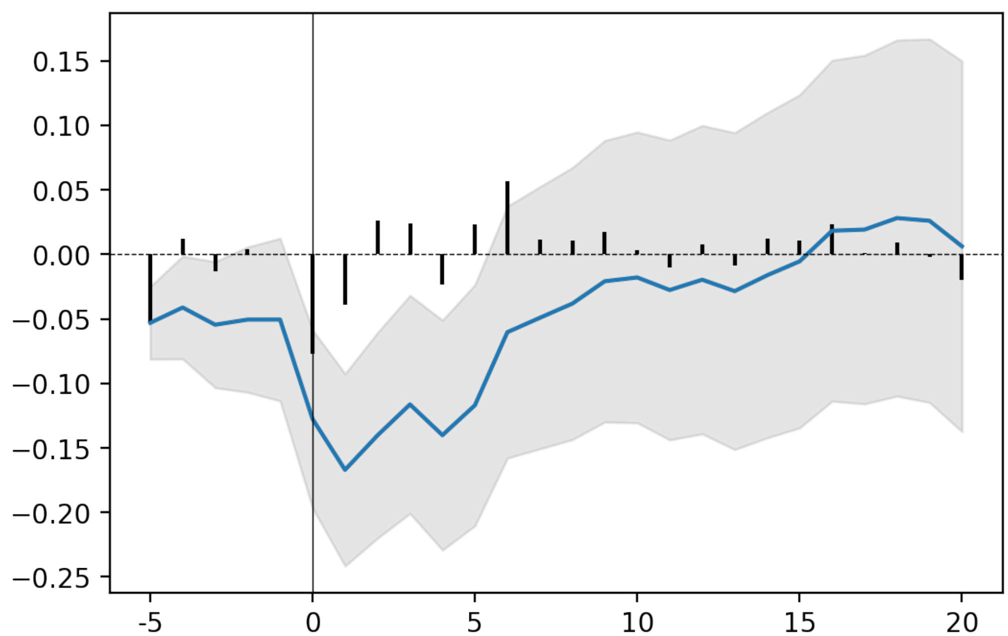
See the documentation on this FamaFrench_3factor function for more details.

See also other models function. You can even set your own modelisation functions

1. Display results:

   - In a plot:

```
event.plot(AR=True)
plt.show() # use standard matplotlib function to display the plot
```



grey area: confidence interval (here at 90%); blue line: CAAR; black bars: AR

Note: You can remove the confidence interval (set *CI = False*) or change its level of confidence (set *confidence = .95* for a confidence interval at 95%). By default AR are note display, but by setting *AR* to *True* you can add them to the plot.

See the documentation on this plot function for more details.

   - Or in a table:

```
event.results(decimals=[3,5,3,5,2,2])
```

Note: Stars are added automatically to represent the level of significance (Significance level: *** at 99%, ** at 95%, * at 90%). You can remove stars by setting *stars* parameter at *False*.

decimals is a list of integer setting for each column (except index) the rounding decimal. You can also set one integer (e.g. *decimals = 3*) if you want all columns to be rounded the same.

See the documentation on this results function for more details.

4.1.3 Example 2: A sample of events

4.2 API

The package implements two classes `Single` and `Multiple` to compute event studies respectively on single events (with measurement such as Abnormal Returns (AR) and Cumulative Abnormal Returns (CAR)) and on an aggregate of events (with measurement such as Average Abnormal Returns (AAR) and Cumulative Abnormal Returns (CAAR)).

The second class (`Multiple`) rely on the first one (`Single`) as it basically performs a loop of single event studies and then aggregate them.

4.2.1 Single Class

Event Study core object. Implement the classical event study methodology[1] for a single event. This implementation heavily rely on the work of MacKinlay[2].

References

Run the event study

`eventstudy.Single.__init__`	Low-level way of runing an event study.
`eventstudy.Single.market_model`	Modelise returns with the market model.
`eventstudy.Single.constant_mean`	Modelise returns with the constant mean model.
`eventstudy.Single.FamaFrench_3factor`	Modelise returns with the Fama-French 3-factor model.

eventstudy.Single.__init__

`Single.__init__`(*model_func*, *model_data: dict*, *event_window: tuple = (-10, 10)*, *estimation_size: int = 300*, *buffer_size: int = 30*, *keep_model: bool = False*)

Low-level way of runing an event study. Prefer the simpler use of model methods.

[1] Fama, E. F., L. Fisher, M. C. Jensen, and R. Roll (1969). "The Adjustment of Stock Prices to New Information". In: International Economic Review 10.1, pp. 1–21.

[2] Mackinlay, A. (1997). "Event Studies in Economics and Finance". In: Journal of Economic Literature 35.1, p. 13.

Parameters

- **model_func** – Function computing the modelisation of returns.
- **model_data** (*dict*) – Dictionary containing all parameters needed by *model_func*.
- **event_window** (*tuple, optional*) – Event window pre (*T2*) and post-event (*T3*) lags around the event date (*0*), by default (-10, +10)
- **estimation_size** (*int, optional*) – Size of the estimation for the modelisation of returns [T0,T1], by default 300
- **buffer_size** (*int, optional*) – Size of the buffer window [T1,T2], by default 30
- **keep_model** (*bool, optional*) – If true *model_func* will return the model which will be accessible through the class attributes EventStudy.model, by default False

See also:

`To()`

`EventStudy.market_model()` the market model.

`EventStudy.FamaFrench_3factor()` the Fama-French 3-factor model.

`EventStudy.constant_mean()` the constant mean model.

Example

Run an event study based on : .. the *market_model* function, .. given values for security and market returns, .. and default parameters

```
>>> from ev.models import market_model
>>> event = EventStudy(
...     market_model,
...     {'security_returns':[0.032,-0.043,...], 'market_returns':[0.012,-0.04,...
↪]}
... )
```

eventstudy.Single.market_model

classmethod Single.**market_model**(*security_ticker: str, market_ticker: str, event_date: numpy.datetime64, event_window: tuple = (-10, 10), estimation_size: int = 300, buffer_size: int = 30, keep_model: bool = False, **kwargs*)

Modelise returns with the market model.

Parameters

- **security_ticker** (*str*) – Ticker of the security (e.g. company stock) as given in the returns imported.
- **market_ticker** (*str*) – Ticker of the market (e.g. market index) as given in the returns imported.
- **event_date** (*np.datetime64*) – Date of the event in numpy.datetime64 format.
- **event_window** (*tuple, optional*) – Event window pre (*T2*) and post-event (*T3*) lags around the event date (*0*), by default (-10, +10)

- **estimation_size** (*int, optional*) – Size of the estimation for the modelisation of returns [T0,T1], by default 300
- **buffer_size** (*int, optional*) – Size of the buffer window [T1,T2], by default 30
- **keep_model** (*bool, optional*) – If true *model_func* will return the model which will be accessible through the class attributes EventStudy.model, by default False
- ****kwargs** – Additional keywords have no effect but might be accepted to avoid freezing if there are not needed parameters specified.

See also:

`To()`

EventStudy.FamaFrench_3factor() the Fama-French 3-factor model.

EventStudy.constant_mean() the constant mean model.

Example

Run an event study for the Apple company for the announcement of the first iphone, based on the market model with the S&P500 index as a market proxy.

```
>>> event = EventStudy.market_model(
...     security_ticker = 'AAPL',
...     market_security = 'SPY',
...     event_date = np.datetime64('2007-01-09'),
...     event_window = (-5,+20)
... )
```

eventstudy.Single.constant_mean

classmethod Single.**constant_mean**(*security_ticker*, *event_date: numpy.datetime64*, *event_window: tuple = (-10, 10)*, *estimation_size: int = 300*, *buffer_size: int = 30*, *keep_model: bool = False*, ***kwargs*)

Modelise returns with the constant mean model.

Parameters

- **security_ticker** (*str*) – Ticker of the security (e.g. company stock) as given in the returns imported.
- **event_date** (*np.datetime64*) – Date of the event in numpy.datetime64 format.
- **event_window** (*tuple, optional*) – Event window pre (*T2*) and post-event (*T3*) lags around the event date (*0*), by default (-10, +10)
- **estimation_size** (*int, optional*) – Size of the estimation for the modelisation of returns [T0,T1], by default 300
- **buffer_size** (*int, optional*) – Size of the buffer window [T1,T2], by default 30
- **keep_model** (*bool, optional*) – If true *model_func* will return the model which will be accessible through the class attributes EventStudy.model, by default False
- ****kwargs** – Additional keywords have no effect but might be accepted to avoid freezing if there are not needed parameters specified. For example, if market_ticker is specified.

See also:

`To()`

`EventStudy.market_model()` the market model.

`EventStudy.FamaFrench_3factor()` the Fama-French 3-factor model.

Example

Run an event study for the Apple company for the announcement of the first iphone, based on the constant mean model.

```
>>> event = EventStudy.constant_mean(
...     security_ticker = 'AAPL',
...     event_date = np.datetime64('2007-01-09'),
...     event_window = (-5,+20)
... )
```

eventstudy.Single.FamaFrench_3factor

classmethod `Single.`**`FamaFrench_3factor`**(*security_ticker*, *event_date: numpy.datetime64*, *event_window: tuple = (-10, 10)*, *estimation_size: int = 300*, *buffer_size: int = 30*, *keep_model: bool = False*, ***kwargs*)

Modelise returns with the Fama-French 3-factor model. The model used is the one developped in Fama and French (1992)[1].

Parameters

- **security_ticker** (*str*) – Ticker of the security (e.g. company stock) as given in the returns imported.
- **event_date** (*np.datetime64*) – Date of the event in numpy.datetime64 format.
- **event_window** (*tuple, optional*) – Event window pre (*T2*) and post-event (*T3*) lags around the event date (*0*), by default (-10, +10)
- **estimation_size** (*int, optional*) – Size of the estimation for the modelisation of returns [T0,T1], by default 300
- **buffer_size** (*int, optional*) – Size of the buffer window [T1,T2], by default 30
- **keep_model** (*bool, optional*) – If true *model_func* will return the model which will be accessible through the class attributes EventStudy.model, by default False
- ****kwargs** – Additional keywords have no effect but might be accepted to avoid freezing if there are not needed parameters specified. For example, if market_ticker is specified.

See also:

`To()`

`EventStudy.market_model()` the market model.

`EventStudy.constant_mean()` the constant mean model.

[1] Fama, E. F. and K. R. French (1992). "The Cross-Section of Expected Stock Returns". In: The Journal of Finance 47.2, pp. 427–465.

Example

Run an event study for the Apple company for the announcement of the first iphone, based on the Fama-French 3-factor model.

```
>>> event = EventStudy.FamaFrench_3factor(
...     security_ticker = 'AAPL',
...     event_date = np.datetime64('2007-01-09'),
...     event_window = (-5,+20)
... )
```

References

Import data

eventstudy.Single.import_FamaFrench	Import returns from csv file to EventStudy Class parameter.
eventstudy.Single.import_returns	Import returns from csv file to EventStudy Class parameter.
eventstudy.Single.import_returns_from_API	

eventstudy.Single.import_FamaFrench

classmethod `Single.`**`import_FamaFrench`**(*path: str*, ***, *rescale_factor: bool = True*, *date_format: str = '%Y%m%d'*)

Import returns from csv file to EventStudy Class parameter. Once imported, the returns are shared among all EventStudy instance.

Parameters

- **path** (*str*) – Path to the returns' csv file
- **rescale_factor** (*bool, optional*) – Divide by 100 the factor provided, by default True, Fama-French factors are given in percent on Kenneth R. French website.
- **date_format** (*str, optional*) – Format of the date provided in the csv file, by default "%Y-%m-%d". Refer to datetime standard library for more details date_format: https://docs.python.org/2/library/datetime.html#strftime-strptime-behavior

eventstudy.Single.import_returns

classmethod `Single.`**`import_returns`**(*path: str*, ***, *is_price: bool = False*, *log_return: bool = True*, *date_format: str = '%Y-%m-%d'*)

Import returns from csv file to EventStudy Class parameter. Once imported, the returns are shared among all EventStudy instance.

Parameters

- **path** (*str*) – Path to the returns' csv file
- **is_price** (*bool, optional*) – Specify if the file contains price (True) or returns (False), by default False. If set at True, the function will convert prices to returns.

- **log_return** (*bool, optional*) – Specify if returns must be computed as log returns (True) or percentage change (False), by default True. Only used if ‘is_price‘is set to True.
- **date_format** (*str, optional*) – Format of the date provided in the csv file, by default “%Y-%m-%d”. Refer to datetime standard library for more details date_format: https://docs.python.org/2/library/datetime.html#strftime-strptime-behavior

eventstudy.Single.import_returns_from_API

classmethod Single.**import_returns_from_API**()

Retrieve results

eventstudy.Single.plot	Plot the event study result.
eventstudy.Single.results	Give event study result in a table format.

eventstudy.Single.plot

Single.**plot**(**, AR=False, CI=True, confidence=0.9*)

Plot the event study result.

Parameters

- **AR** (*bool, optional*) – Add to the figure a bar plot of AR, by default False
- **CI** (*bool, optional*) – Display the confidence interval, by default True
- **confidence** (*float, optional*) – Set the confidence level, by default 0.90

Returns Plot of CAR and AR (if specified).

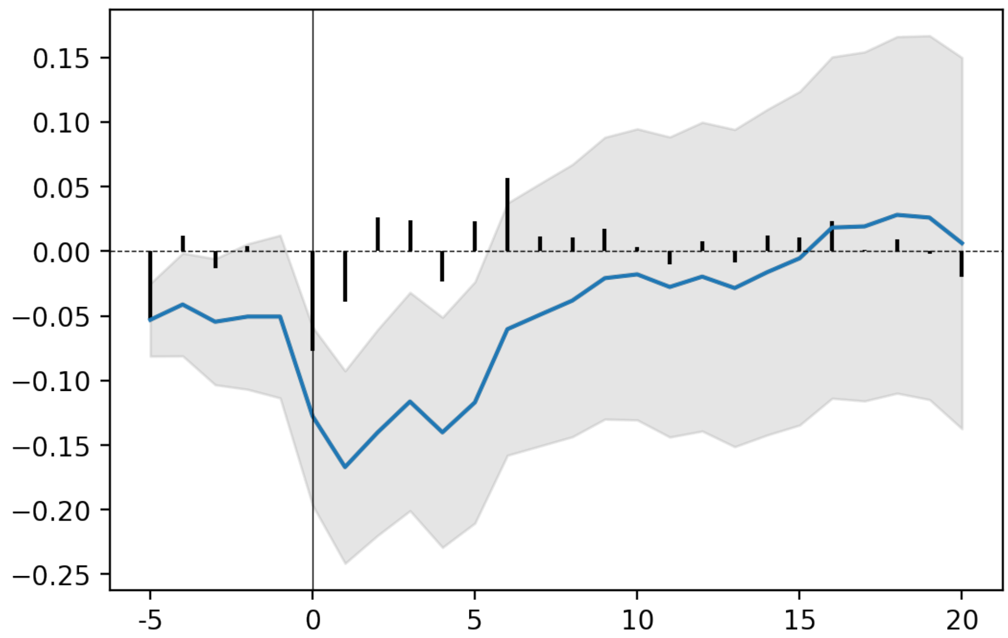
Return type matplotlib.figure

Note: The function return a fully working matplotlib function. You can extend the figure and apply new set-up with matplolib’s method (e.g. savefig).

Example

Plot CAR (in blue) and AR (in black), with a confidence interval of 95% (in grey).

```
>>> event = EventStudy.market_model(
...     security_ticker = 'AAPL',
...     market_ticker = 'SPY',
...     event_date = np.datetime64('2007-01-09'),
...     event_window = (-5,+20)
... )
>>> event.plot(AR = True, confidence = .95)
```



eventstudy.Single.results

`Single.results(stars: bool = True, decimals=3)`

Give event study result in a table format.

Parameters

- **stars** (*bool, optional*) – Add stars to CAR value based on significance of p-value, by default True
- **decimals** (*int or list, optional*) – Round the value with the number of decimal specified, by default 3. *decimals* can either be an integer, in this case all value will be round at the same decimals, or a list of 6 decimals, in this case each columns will be round based on its respective number of decimal.

Note: When *stars* is set as True, CAR's are converted to string type. To make further computation on CARs possible set *stars* to False.

Returns AR and AR's variance, CAR and CAR's variance, T-stat and P-value, for each T in the event window.

Return type pandas.DataFrame

Note: The function return a fully working pandas DataFrame. All pandas method can be used on it, especially exporting method (to_csv, to_excel,. . .)

Example

Get results of a market model event study, with specific number of decimal for each column:

```
>>> event = EventStudy.market_model(
...     security_ticker = 'AAPL',
...     market_ticker = 'SPY',
...     event_date = np.datetime64('2007-01-09'),
...     event_window = (-5,+5)
... )
>>> event.results(decimals = [3,5,3,5,2,2])
```

	AR	Variance AR	CAR	Variance CAR	T-stat	P-value
-5	-0.053	0.00048	-0.053 **	0.00048	-2.42	0.01
-4	0.012	0.00048	-0.041 *	0.00096	-1.33	0.09
-3	-0.013	0.00048	-0.055 *	0.00144	-1.43	0.08
-2	0.004	0.00048	-0.051	0.00192	-1.15	0.13
-1	0	0.00048	-0.051	0.00241	-1.03	0.15
0	-0.077	0.00048	-0.128 **	0.00289	-2.37	0.01
1	-0.039	0.00048	-0.167 ***	0.00337	-2.88	0
2	0.027	0.00048	-0.14 **	0.00385	-2.26	0.01
3	0.024	0.00048	-0.116 **	0.00433	-1.77	0.04
4	-0.024	0.00048	-0.14 **	0.00481	-2.02	0.02
5	0.023	0.00048	-0.117 *	0.00529	-1.61	0.05

Note: Significance level: *** at 99%, ** at 95%, * at 90%

4.2.2 Multiple Class

Run the event study

eventstudy.Multiple.__init__	Initialize self.
eventstudy.Multiple.from_csv	
eventstudy.Multiple.from_list	
eventstudy.Multiple.from_text	
eventstudy.Multiple.error_report	

eventstudy.Multiple.__init__

Multiple.**__init__**(*sample*, *errors=None*)

Initialize self. See help(type(self)) for accurate signature.

eventstudy.Multiple.from_csv

classmethod Multiple.**from_csv**(*path*, *event_study_model*, *event_window: tuple = (-10, 10)*, *estimation_size: int = 300*, *buffer_size: int = 30*, *keep_model: bool = False*, ***, *date_format: str = '%Y%m%d'*, *ignore_errors: bool = True*)

eventstudy.Multiple.from_list

classmethod Multiple.**from_list**(*event_list*, *event_study_model*, *event_window: tuple = (-10, 10)*, *estimation_size: int = 300*, *buffer_size: int = 30*, ***, *keep_model: bool = False*, *ignore_errors: bool = True*)

eventstudy.Multiple.from_text

classmethod Multiple.**from_text**(*text*, *event_study_model*, *event_window: tuple = (-10, 10)*, *estimation_size: int = 300*, *buffer_size: int = 30*, ***, *date_format: str = '%Y-%m-%d'*, *keep_model: bool = False*, *ignore_errors: bool = True*)

eventstudy.Multiple.error_report

Multiple.**error_report**()

Import data

Note: Returns and factor data are directly imported at the single event study level.

eventstudy.Single.import_FamaFrench	Import returns from csv file to EventStudy Class parameter.
eventstudy.Single.import_returns	Import returns from csv file to EventStudy Class parameter.
eventstudy.Single.import_returns_from_API	

Retrieve results

eventstudy.Multiple.plot	Plot the event study result.
eventstudy.Multiple.results	Give event study result in a table format.
eventstudy.Multiple.get_CAR_dist	Give CARs' distribution descriptive statistics in a table format.
eventstudy.Multiple.sign_test	
eventstudy.Multiple.rank_test	

eventstudy.Multiple.plot

`Multiple.plot` (*, *AAR=False*, *CI=True*, *confidence=0.9*)

Plot the event study result.

Parameters

- **AAR** (*bool, optional*) – Add to the figure a bar plot of AAR, by default False
- **CI** (*bool, optional*) – Display the confidence interval, by default True
- **confidence** (*float, optional*) – Set the confidence level, by default 0.90

Returns Plot of CAAR and AAR (if specified).

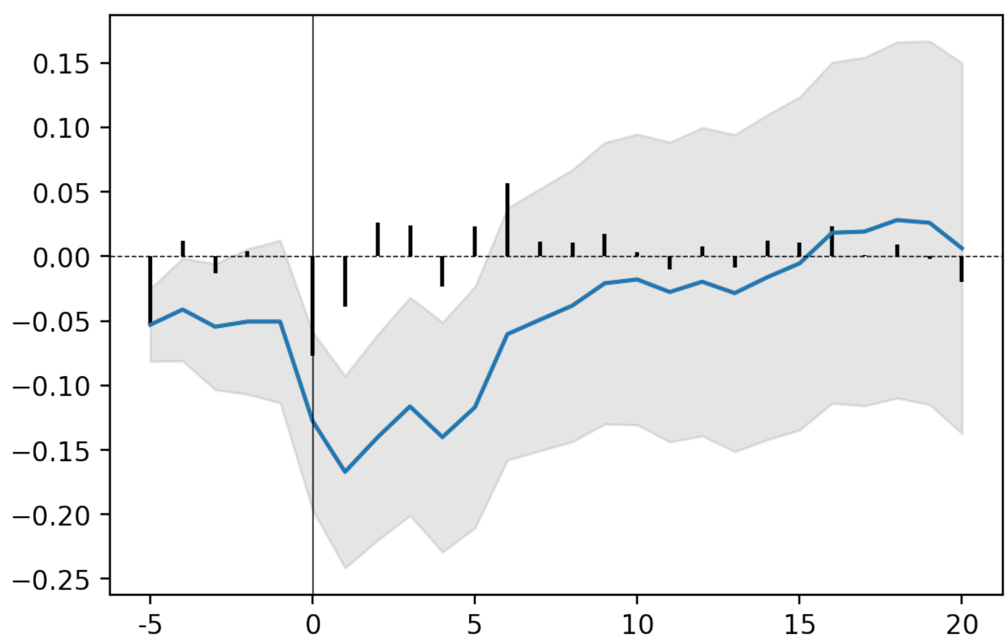
Return type matplotlib.figure

Note: The function return a fully working matplotlib function. You can extend the figure and apply new set-up with matplolib's method (e.g. savefig).

Example

Plot CAR (in blue) and AR (in black), with a confidence interval of 95% (in grey).

```
>>> events = es.Sample.from_csv(
...     'AAPL_10K.csv',
...     es.Single.FamaFrench_3factor,
...     event_window = (-5,+5),
...     date_format = '%d/%m/%Y'
... )
>>> events.plot(AR = True, confidence = .95)
```



eventstudy.Multiple.results

`Multiple.results` (*stars: bool = True*, *decimals=3*)

Give event study result in a table format.

Parameters

- **stars** (*bool, optional*) – Add stars to CAR value based on significance of p-value, by default True
- **decimals** (*int or list, optional*) – Round the value with the number of decimal specified, by default 3. *decimals* can either be an integer, in this case all value will be round at the same decimals, or a list of 6 decimals, in this case each columns will be round based on its respective number of decimal.

Note: When *stars* is set as True, CAR's are converted to string type. To make further computation on CARs possible set *stars* to False.

Returns AAR and AAR's variance, CAAR and CAAR's variance, T-stat and P-value, for each T in the event window.

Return type pandas.DataFrame

Note: The function return a fully working pandas DataFrame. All pandas method can be used on it, especially exporting method (to_csv, to_excel,. . .)

Example

Get results of a market model event study on a sample of events (Apple Inc. 10-K release) imported from a csv, with specific number of decimal for each column:

```
>>> events = es.Sample.from_csv(
...     'AAPL_10K.csv',
...     es.Single.FamaFrench_3factor,
...     event_window = (-5,+5),
...     date_format = '%d/%m/%Y'
... )
>>> events.results(decimals = [3,5,3,5,2,2])
```

	AAR	Variance AAR	CAAR	Variance CAAR	T-stat	P-value
-5	-0	3e-05	-0.0	3e-05	-0.09	0.47
-4	-0.002	3e-05	-0.003	5e-05	-0.35	0.36
-3	0.009	3e-05	0.007	8e-05	0.79	0.22
-2	0.003	3e-05	0.01	0.0001	1.03	0.15
-1	0.008	3e-05	0.018 *	0.00013	1.61	0.05
0	-0	3e-05	0.018 *	0.00015	1.46	0.07
1	-0.006	3e-05	0.012	0.00018	0.88	0.19
2	0.006	3e-05	0.017	0.0002	1.22	0.11
3	0	3e-05	0.018	0.00023	1.17	0.12
4	-0.007	3e-05	0.011	0.00025	0.69	0.24
5	0.001	3e-05	0.012	0.00028	0.72	0.24

Note: Significance level: *** at 99%, ** at 95%, * at 90%

eventstudy.Multiple.get_CAR_dist

`Multiple.get_CAR_dist(decimals=3)`

Give CARs' distribution descriptive statistics in a table format.

Parameters **decimals** (*int or list, optional*) – Round the value with the number of decimal specified, by default 3. *decimals* can either be an integer, in this case all value will be round at the same decimals, or a list of 6 decimals, in this case each columns will be round based on its respective number of decimal.

Returns CARs' descriptive statistics

Return type pandas.DataFrame

Note: The function return a fully working pandas DataFrame. All pandas method can be used on it, especially exporting method (to_csv, to_excel,. . .)

Example

Get CARs' descriptive statistics of a market model event study on a sample of events (Apple Inc. 10-K release) imported from a csv, with specific number of decimal for each column:

```
>>> events = es.Sample.from_csv(
...     'AAPL_10K.csv',
...     es.Single.FamaFrench_3factor,
...     event_window = (-5,+5),
...     date_format = '%d/%m/%Y'
... )
>>> events.get_CAR_dist(decimals = 4)
```

	Mean	Variance	Kurtosis	Min	Quantile 25%	Quantile 50%	Quantile 75%	Max
-5	-0	0.001	0.061	-0.052	-0.014	0.001	0.015	0.047
-4	-0.003	0.001	0.247	-0.091	-0.022	0.003	0.015	0.081
-3	0.007	0.002	0.532	-0.082	-0.026	0.006	0.027	0.139
-2	0.01	0.002	-0.025	-0.088	-0.021	0.002	0.033	0.115
-1	0.018	0.003	-0.065	-0.091	-0.012	0.02	0.041	0.138
0	0.018	0.003	-0.724	-0.084	-0.012	0.012	0.057	0.128
1	0.012	0.004	-0.613	-0.076	-0.024	0.003	0.059	0.143
2	0.017	0.005	-0.55	-0.117	-0.026	0.024	0.057	0.156
3	0.018	0.005	0.289	-0.162	-0.032	0.027	0.057	0.17
4	0.011	0.007	2.996	-0.282	-0.039	0.035	0.052	0.178
5	0.012	0.008	1.629	-0.266	-0.05	0.035	0.064	0.174

Note: Significance level: *** at 99%, ** at 95%, * at 90%

eventstudy.Multiple.sign_test

`Multiple.`**`sign_test`**(*sign='positive'*, *confidence=0.9*)

eventstudy.Multiple.rank_test

`Multiple.`**`rank_test`**(*confidence*)

PYTHON MODULE INDEX

e

INDEX

Symbols

C

E

F

G

I

M

P

R

S