Using explanations

The description and explanation strings on aggregate objects

Stephen J. Mildenhall

2026-11-01

Many facts about a computed object are reported twice, once in a table and once in words: whether the grid is good enough, why the bucket has its size, how thick the tail is, and what the reinsurance cedes. The words come as pairs of read-only string properties sharing a stem[1]. The string `<stem>_description` is one line or a few aligned lines, for scanning. The string `<stem>_explanation` is prose: what was checked or chosen, against what, and what to do about it. Wherever one half exists the other does too. Both are computed on access, so they follow the object as it changes. The short forms of validation, tail, and reinsurance also appear in the attribute `info`.

Validation compares the severity and aggregate mean, coefficient of variation (CV), and skewness on the realized grid with their analytic values. A relative error above `validation_eps` (10^{-4} by default) fails a mean, above ten times that a CV, and above a hundred times that a skewness. For the severity and for the aggregate only the lowest-order failure is reported, since a wrong mean makes the higher moments uninformative.

The examples below use objects built from these programs.

```
agg Tour 20 claims sev lognorm 100 cv 1.5 occurrence net of 500 xs 100 poisson
port Two
  agg A 10 claims sev gamma 50 cv 0.5 poisson
  agg B 5 claims sev lognorm 100 cv 1 poisson
agg LT.Basic elt [[1 .1 100] [2 .1 40] [3 .1 80]]
pnl PL 100 premium less agg PL.B 10 claims sev lognorm 8 cv 1.5 poisson
```

```
import pandas as pd
from aggregate import build, qd, explain_validation, __version__
from aggregate import (Aggregate, Portfolio, PnL, Severity, Frequency,
                       Distortion)
from aggregate.bivariate import BivariateAggregate
from style import wprint, GT, figures
figures()
print(__version__)
```

```
1.58.0
```

Description and explanation pairs by class

The table marks, for each class, the stems that have a pair. It is computed from the classes themselves, so it stays current. A `Distortion` has none of the pairs.

[1] From aggregate 1.0, and the `elt` pair from 1.21.0

```
stems = ['validation', 'bs', 'tail', 'sharpen', 'reins', 'elt', 'construction']
classes = [Aggregate, Portfolio, BivariateAggregate, PnL, Severity, Frequency,
           Distortion]
has = {s: ['x' if f'{s}_description' in dir(c) else '' for c in classes]
       for s in stems}
names = pd.Index([c.__name__ for c in classes], name='class')
GT(pd.DataFrame(has, index=names))
```

class	validation	bs	tail	sharpen	reins	elt	construction
Aggregate	x	x	x	x	x	x	
Portfolio	x	x	x	x	x		
BivariateAggregate	x	x	x				
PnL							x
Severity			x				
Frequency			x				
Distortion							

Validation against the analytic moments

The string `validation_description` is the verdict. Under reinsurance it has two clauses, one for the gross subject against its analytic moments and one for the net against the exact image of the subject on the grid (Using validation). The function `explain_validation` returns the words for the flags in the attribute `valid`. Flags alone cannot separate a subject from a view, so on a reinsured object it gives only the first clause.

```
a = build('agg Tour 20 claims sev lognorm 100 cv 1.5 '
          'occurrence net of 500 xs 100 poisson')
print(a.validation_description)
print(explain_validation(a.valid))
wprint(a.validation_explanation)
```

```
reinsurance; subject not unreasonable; net not unreasonable
reinsurance; subject not unreasonable
Validation of the realized grid against the analytic moments: reinsurance;
subject not unreasonable; net not unreasonable. Each of mean, CV and skewness is
compared for severity and for the aggregate, and a relative error above 0.0001
fails; only the lowest-order failure is reported, since a mean that is wrong
makes the higher moments uninformative. The reported view is net or ceded. The
subject (gross) object underneath is checked against its analytic moments, and
the cession against the exact image of the subject on its grid.
```

The choice of bucket and window

The string `bs_description` gives the grid and any clipped tail, and `bs_explanation` gives the sizing methods that applied, the winner, and any single big jump reach the window misses. The table behind them is `bs_window_df`, described in Using FFT windows and, with the tail classes, in Using tail classes.

```
wprint(a.bs_description)
wprint(a.bs_explanation)
```

```
moment grid: bs=1/2, log2=16, window [0, 32,768] of width 32,768; clips
~5.79e-08 of the tail (raise log2 to 19)
The aggregate is severity-driven, [0, inf), subexponential right tail. Log2 is
16. Of the methods that applied (moment 29,492, sbj 180,738), the moment method
won (window width 29,492) -- the 3-moment method-of-moments window. It has a
natural lower support bound of 0. The recommended x_min is 0 and the x_max that
follows is 32,768, so the window is [0, 32,768] of width 32,768. The heavy right
tail reaches 180,738, past the grid x_max 32,768 (~5.79e-08 of the mass clipped,
a reported deficit not normalized); the analysis suggests increasing log2 to 19.
```

Reinsurance terms and cessions

The string `reins_description` restates the reinsurance terms, and `reins_explanation` adds their effect on expected loss, stage by stage. For `Tour` the occurrence layer 500 xs 100 cedes 727 of the gross expected loss of 2,000, or 36.4 percent.

```
wprint(a.reins_description)
wprint(a.reins_explanation)
```

```
Net of 100% share of 500 xs 100 per occurrence
Net of 100% share of 500 xs 100 per occurrence. Per occurrence, gross expected
loss 2,000 splits into 727.203 ceded (36.4%) and 1,272.8 net.
```

The sharpen pair

Before the method `sharpen()` is called, the string `sharpen_description` is `Sharpen: not run.` After the call it gives the probe and its outcome (Using `sharpen`). The explanation runs to a page, with the definition of the score and of the search. It ends with the reason the grid moved or stayed, the part shown here.

```
print(a.sharpen_description)
a.sharpen()
wprint(a.sharpen_description)
wprint(a.sharpen_explanation.rsplit('Here: ', 1)[1])
```

```
Sharpen: not run.
Sharpen: 6 cells in 0.14s, best score 0.285 vs 0.285 at the center, target 0.5.
Kept bs 1/2, log2 16.
best sound score at the current grid size or smaller, and it meets the target,
so log2 was not grown. No neighbor was enough better to justify moving, so the
grid was kept.
```

Severity and frequency tails

A `Severity` and a `Frequency` have only the tail pair. The components of an aggregate are available as `sevs`, a list of severities, and `frequency`.

```
wprint(a.sevs[0].tail_explanation)
wprint(a.frequency.tail_explanation)
```

```
Lognorm severity [0, inf), subexponential right tail. Declared support:
unlimited. The right tail is thick (subexponential): the aggregate inherits it
by the single big jump.
Poisson frequency with a super-exponential count tail. E[N] = 20, SD(N) =
4.47214. It is Poisson-dispersed (var / mean = 1). The count law is log-concave,
so the aggregate tail follows the severity.
```

The tail and grid pairs of a Portfolio

On a `Portfolio` the tail pair gives the unit that drives the tail of the total, and the grid pair describes the one grid all the units share. In `Two` the lognormal unit `B` drives the subexponential tail.

```
p = build('port Two agg A 10 claims sev gamma 50 cv 0.5 poisson '
          'agg B 5 claims sev lognorm 100 cv 1 poisson')
wprint(p.tail_description)
wprint(p.tail_explanation)
wprint(p.bs_description)
```

```
aggregate tail           subexponential (driver: B)
Portfolio aggregate tail is subexponential (worst-of under independence, driven
by B). Units -- A: exponential; B: subexponential.
portfolio grid: bs=1/2, log2=16, window [0, 32,768] of width 32,768
```

The elt pair for loss tables

An `elt` or `yelt` aggregate (Using `elt`, Using `yelt`) has the `elt` pair. The description is a few aligned lines with the table, the event rate, the grid, and the checks. The explanation is three paragraphs, the first of which describes the model in words. On any other aggregate both strings are empty.

```
e = build('agg LT.Basic elt [[1 .1 100] [2 .1 40] [3 .1 80]]')
wprint(e.elt_description)
wprint(e.elt_explanation.split('\n\n')[0])
print(repr(a.elt_description), repr(a.elt_explanation))
```

```
loss table               ELT3, 3 events from 3 rows, 1 unit
event rate               0.3, mean annual loss 22
grid                     bs 20, log2 5
checks                   13 pass
The annual loss is compound Poisson with event rate 0.3 and severity the
rate-weighted mixture of the event totals, so the mean annual loss is 22. An
event's unit losses are summed into one event total before any occurrence
treaty, and equal totals are combined: the 3 events with a positive rate give 3
severity atoms.
'' ''
```

The construction pair of a PnL

A `PnL` (Using `pnl` and `xpnl`) has the `construction` pair. The description is one paragraph with the route, the source, and the groups. The explanation opens with the description and adds the booking and a replay of the constructor call, with placeholders for function legs.

```
q = build('pnl PL 100 premium less '
          'agg PL.B 10 claims sev lognorm 8 cv 1.5 poisson')
wprint(q.construction_explanation)
```

```
Plain pnl, consolidated face: one sell group over the engine's own density
(Aggregate); consideration 100; scenario (κ) ladder (single shared source).
Booking: sell -- +consideration, -obligation; loss-basis LAE is stochastic rate
* x (the loss is on-source). For a reinsurance-bearing engine the density here
is the net -- what comes out of the engine.

Replay (function legs as <fn> placeholders):
    PnL(name='PL',
        source=<GridDistribution PL>,
        groups=[Group('PL', 'sell', consideration=[Leg('Premium', 100.0,
        kind='premium')], obligation=[Leg('Loss', <fn>, kind='loss')])],
        result_name='margin')
```