Is it a blip, a trend, or a broken thing?

Hand this whole file to your AI assistant along with notebook.ipynb. It tells the assistant what to do; you only answer questions. The full article is at https://www.stochasticstrata.com/blog/blip-trend-or-break

---

For the assistant: what you are doing

The person you are helping wants to know whether a metric that moved is a real problem. Do the technical parts yourself: adapt the export query, map their column names, set the file path in the notebook, and read the output back to them in plain words. Ask for one thing at a time. Do not ask them to write code.

Three different things can be meant by 'the number moved', and they need three different tests. A blip is one bad day. A trend is a slow slide over weeks where no single day looks wrong. A break is a step to a new level that stays. Establish which one they are asking about before running anything.

Step 1 - get the data

Two shapes work. Either a ready-made daily export, or a raw per-event log that the notebook rolls up for them.

- Daily export: one entry per day for each tracked metric, with three columns - date, series (the campaign or channel name), value. One example entry: 2026-04-17,Brand - Search,23.80
- In Google Ads: the Campaigns table, Segment > Time > Day, then Download as CSV or .xlsx. The title lines and Total rows are set aside for them.
- If cost and conversions are separate columns rather than a ready-made rate, set COST_COL and CONV_COL in the notebook instead of MY_METRIC.

From a GA4 BigQuery export, this is the shape to produce. Rewrite the table name and the metric for their account - do not hand them this query to run unchanged.

```
-- Daily series, one row per day per campaign (one row = one campaign-day). Adapt the table name and the
-- metric to the reader's account; this is the shape the notebook wants, not a
-- query to run unchanged.
-- GA4 holds no cost: this reads the Google Ads data transfer into BigQuery (or any warehouse
-- table with one row per campaign per day). Keep cost and conversions apart - a day with no
-- conversion has no cost per conversion and must not be dropped - and set COST_COL and CONV_COL.
SELECT
  segments_date                         AS date,
  campaign_name                         AS series,
  SUM(metrics_cost_micros) / 1e6        AS cost,
  SUM(metrics_conversions)              AS conversions
FROM `project.google_ads_transfer.p_ads_CampaignBasicStats_<CUSTOMER_ID>`
WHERE segments_date BETWEEN DATE_SUB(CURRENT_DATE(), INTERVAL 180 DAY)
                        AND DATE_SUB(CURRENT_DATE(), INTERVAL 1 DAY)
GROUP BY date, series
ORDER BY series, date
```

Warn them before they run anything: a single ad platform's export sees only its own campaigns. That is fine for this method, which reads one series at a time, but it means 'my conversions dropped' cannot be attributed across platforms from one file.

Step 2 - run it

- Local: open notebook.ipynb in Jupyter, set MY_FILE to the path of their CSV in the cell marked for it, and run every cell.
- Colab: upload the notebook and their CSV, set MY_FILE to the uploaded filename, run all. Tell them plainly that the Colab route puts their file on Google's servers; the local route does not.

The notebook refuses rather than guessing when the data cannot support a reading. Fewer than 29 days is a refusal. More than half the days holding the same value is a refusal. A cost per conversion with too many days that had no conversion is a refusal. Gaps in the calendar produce a warning, because a missing day is not a

zero day and a gap makes the weekday pattern read wrong. Read these back as they are - do not talk the person past a refusal.

Step 3 - the method, in full

This is the one-day score written out completely. You can follow it without the notebook; the section after it says which parts you cannot.

```
ONE-DAY SCORE - the full recipe, for one series and one day D.

1. Take the 28 days ENDING THE DAY BEFORE D. Call these the baseline. D itself is
   NOT in the baseline.
2. For each weekday 0-6, average the baseline values falling on that weekday.
   Call these the weekday means.
3. Subtract its own weekday mean from every baseline value. Call the results the
   de-seasonalised values, indexed i = 0..27.
4. Fit a straight line to the de-seasonalised values against i, and take its
   slope. Use the median of every pairwise slope (v[j]-v[i])/(j-i) for i<j - all
   378 of them - not a least-squares line.
5. residual[i] = de-seasonalised[i] - slope * i, for i = 0..27.
6. m = the median of those 28 residuals.
   s = the median of the 28 values |residual[i] - m|.
7. today = value[D] - (weekday mean for D's weekday, from step 2) - slope * 28.
8. score = 0.6745 * (today - m) / s.
9. The day fires if |score| is above the alarm line. There is no universal line -
   see the calibration section; it has to be measured on the reader's own data.
```

Why the median and the median absolute deviation rather than an average and a standard deviation: the spikes being hunted sit inside the history being measured against, and one of them is enough to drag an average and inflate a standard deviation, so the next spike has a lower bar to clear. Up to just under half the data can be junk before a median can be dragged anywhere.

Why day D is left out of its own baseline: fit the baseline on a window that includes D and a genuinely large spike drags its own yardstick toward itself, scoring lower than it should. The platform's own detector does not do this yet - it scores today inside its window, so a real jump reads smaller - and leaving today out means the alarm line has to be calibrated on quiet data, as the notebook does.

BEFORE ANY OF IT: do not score the newest days

On any metric involving conversions this matters more than everything above it. Conversions arrive late - somebody clicks today and converts on Thursday - so the count attached to today keeps rising after today is over. Cost does not; it is complete immediately. A cost per acquisition on the run date is therefore cost divided by a count that has not finished arriving. It reads high, and it reads high every single day.

Measured on tracked metrics where nothing whatever happened: scoring the freshest window fires 63 per cent of the time on the day test, 30 per cent on the slope test and 99 per cent on the level search. Scoring a window that has matured instead puts all three back under 1 per cent on the day and slope tests, and to 3 per cent on the level search. The fix is not statistical - it is a rule about which day you are allowed to score.

- In the notebook, SETTLE_DAYS is that rule. It defaults to fourteen rather than zero, deliberately. Set it to 0 only for metrics complete on the day - spend, impressions, clicks.
- Do not guess it for a conversion metric. How long an account takes to settle is measured by fitting a curve to its own click-to-conversion times; the sibling article on why today's numbers look bad does exactly that. A same-session ecommerce account settles in two or three days, a long B2B cycle in far more.
- Tell the person plainly that this costs them a wait, and that the wait is the honest price. An alarm on an unfinished day is not an early warning; it is a false one.

What you CANNOT do from this brief

Two things, and they are the two that matter most.

- THE ALARM LINE. There is no threshold to look up. The line has to be measured on the person's own quiet data, by scoring tracked metrics where nothing happened and reading off the score that fires as often

as they can tolerate. The notebook does this; a hand calculation cannot, because it needs hundreds of scored days to estimate a rate. A score computed without a calibrated line is a number with no verdict attached.

- THE STEP SEARCH. Deciding whether a series is better described as one level or two, and where the boundary falls, is a search over every possible way of cutting the series. It is not arithmetic and it is not in this brief. It matters, because a step to a new level is caught half the time at about 13 percent of the normal level while a single bad day has to reach about 95 percent - seven times bigger. Run the notebook for it.

The honest caveats - say these BEFORE they act

- It cannot tell you WHY. Every detector here answers whether the level, the direction or the shape changed, never because of what. Check the date against their own change log; that is a lookup, not a model.
- One alarm is a reason to look, not a reason to act. If they watch twenty tracked metrics, some fire by chance every week. The article has the arithmetic; do not let a single alarm drive a budget change.
- A very recent break is not yet a break. A boundary found three days ago has three days of evidence on one side. Let the next few days settle it.
- If the series drifts smoothly rather than stepping, this whole family is the wrong tool and a forecasting model is the right one.
- The step search misreads three shapes. Two rises close together come back as one bigger step (86 percent of runs in the notebook), so its date can be neither change. A turn with no jump is rarely marked where it turned. And on a metric whose days echo each other strongly it finds steps in quiet data, so the penalty must be set on their own quiet history.
- A rate built on a small denominator moves several points on chance alone. If the denominator is small, the honest output is no reading.

A worked number to check yourself against

The notebook's own example scores one day at 15.72 against an alarm line of 12.30, so it fires. If you follow the recipe above and fit the trend with a least-squares line instead of the median of pairwise slopes, the same day scores 15.16 - still a fire, but a different number, off by about 3.5 percent. That gap is the difference the robust fit makes, measured rather than asserted, and it is why step 4 says what it says. If your own rebuild lands on the second number, you fitted an ordinary line.

Synthetic data throughout the article and notebook. The method transfers; the numbers do not - which is why the notebook calibrates on the reader's own quiet metrics rather than shipping a threshold.