Maturity Grade: the full algorithm

A freebie from hoiboy.uk.

Hand this whole document to an AI (Claude, ChatGPT, or any coding assistant) and ask it to build the maturity grade into your own trading system. Everything is here: what each number means in plain English, the exact formulas, the settings, and a working reference implementation in Python.

The story behind it, and why it works, is on the blog: hoiboy.uk/posts/maturity-grading-from-backtest-data

This is the real thing I run, not a watered-down version. The only part I keep to myself is my actual trading-system codebase. The grade is below in full, with every term explained plainly, so even if this is new to you it should make sense.

The one-line idea

Every stock gets a letter, A to E, on each of three timeframes (daily, weekly, monthly). **A** means the stock's price swings have been getting BIGGER lately, so it is waking up and moving. **E** means it has gone quiet and flat, basically asleep.

Why measure that? Because you only make money when a stock actually moves. A stock that just sits there is dead money. So this grade answers one simple question at any moment: is this stock waking up, or is it asleep? You then read the three letters together to see where the stock is in its cycle (the blog post explains that part).

What you feed it (the inputs)

For each stock you need its price history as a table of **bars**. One bar = one time period, holding four prices for that period: the **O**pen, **H**igh, **L**ow and **C**lose, plus the **V**olume (how many shares traded). That table is called OHLCV. Sort it oldest-first.

You run the whole thing three times, once per timeframe: daily bars, weekly bars, monthly bars. Each run gives the stock one letter for that timeframe. The grade only looks at the High, Low and Close. It ignores Volume.

Step 1: turn each bar into three simple measurements

For every bar we work out three numbers (call them "features"). One golden rule: each number is worked out using ONLY that bar and the bars BEFORE it, never the bars after. In a real trade you cannot see the future, so the grade must not either. (The jargon for this is "causal" — no peeking ahead.)

Measurement 1, `atr_pct`: how much is this stock moving, as a % of its price?

Start with the "True Range" (TR) of a bar, which is just how far the price travelled that bar. It is the biggest of these three:

```
TR = the biggest of:
       high - low                 (the bar's own high-to-low range)
       abs(high - prev_close)     (the jump up from the previous close)
       abs(low  - prev_close)     (the jump down from the previous close)
```

We include the gap from the previous bar's close so an overnight jump still counts. Then **ATR(14)** is the average True Range over the last 14 bars (a smoothed average that leans on the most recent bars; this is the standard "Wilder" method). In plain words: the stock's typical move size lately.

```
atr_pct = ATR(14) / close
```

We divide by the price so a £500 stock and a £5 stock can be compared fairly. A bigger `atr_pct` means the stock is swinging around more than usual, i.e. waking up.

Measurement 2, `breakout_hi`: is the price pushing above its recent ceiling?

Look back over the last `coil` bars and find the highest high in that window. That is the stock's recent ceiling. (We call the window `coil` because a stock often coils up under a ceiling before it breaks out.) Then:

```
breakout_hi = (close / recent_ceiling) - 1
```

That is simply "how far above or below the recent ceiling is today's close, as a fraction". A positive number means price is breaking out above where it had been stuck. We take the ceiling from the bars BEFORE today, so a bar can never be its own ceiling.

Measurement 3, `mom_atr`: how big has the recent move been, measured in 'normal moves'?

First, what "return" means here: it is the change in the STOCK'S PRICE over a window, written as a percentage. It is the stock going up or down. It is NOT your trading profit or loss. Example: if the close was 100 a while back and it is 120 now, the return is +20%.

We take the return over the last `fast` bars, then divide it by `atr_pct`:

```
mom_atr = ( (close / close_`fast`_bars_ago) - 1 ) / atr_pct
```

Dividing by `atr_pct` rescales the move into "how many of this stock's normal-sized moves have stacked up". A big `mom_atr` means the stock has travelled a long way compared with its usual wiggle, i.e. strong momentum.

Two window settings appeared above. `coil` is how many bars back we look for the ceiling, and `fast` is how many bars back we measure the recent move. Their exact sizes per timeframe are in the table in Step 2.

Step 2: combine the three measurements into one score

The three measurements are on different scales, so before adding them we put each on a shared "how unusual is this?" scale called a z-score (explained in Step 3). Then we add them up, each with a + or − weight:

```
score = sum of ( weight x z-score of each measurement )
```

Here are the weights and the two window settings, per timeframe:

timeframe	atr_pct	breakout_hi	mom_atr	coil (bars)	fast (bars)
daily	+1	not used	not used	30	20
weekly	+1	not used	not used	15	10
monthly	+1	−1	+1	9	10

How to read the table:

- The three middle columns are the WEIGHTS. `+1` means "add this measurement in", `−1` means "subtract it", "not used" means that timeframe ignores it.
- **coil (bars)** = how many bars back to look for the recent ceiling (used by `breakout_hi`).
- **fast (bars)** = how many bars back to measure the recent move (used by `mom_atr`).
- The ATR window is always 14 bars, on all three timeframes.

So daily and weekly are simple: the score is just how unusually big the stock's recent moves are (`atr_pct`). Monthly is richer: it also adds momentum (`mom_atr`, weight +1) and subtracts how stretched the price already is above its ceiling (`breakout_hi`, weight −1, so a stock that has already shot far above its recent high scores a little lower, because it may be late). These weights were not picked by hand; they are what came out of the backtesting.

Step 3: the z-score and the "fixed reference" (the important bit)

A z-score answers "how unusual is this number, compared to normal?" To work one out you need two facts about each measurement, gathered from lots of history:

- the **mean** = the plain average value.
- the **standard deviation** (std) = the typical amount values stray from that average (the normal spread).

Then:

```
z-score = (value - mean) / std,   then capped to stay within -4 to +4
```

A z-score of 0 means "dead average". +1 means "one normal step above average", +2 "well above", −1 "below average", and so on. Capping at ±4 stops one freak value from dominating. The z-score is what lets us add `atr_pct`, `breakout_hi` and `mom_atr` together even though they are measured in different units.

Now the key trick. The mean, the std, and the grade cut-offs all come from a **fixed reference** that you work out ONCE, from years of bars across loads of stocks, then freeze. After that you grade any single bar on its own, just by comparing it to those frozen numbers.

Why freeze it? The obvious-but-wrong alternative is to rank each bar against all the other bars you happen to have right now. That breaks in two ways: you cannot grade anything until you already hold the whole pile (so it only works after the fact, useless live), and one stock's grade shifts every time you add or remove another stock. A frozen reference fixes both. The same bar always gets the same grade, live or in a backtest, with the same code.

To build the reference, for each timeframe:

1. Work out the three measurements for a big pile of stocks across years of bars.
2. For each measurement that timeframe uses, save its mean and its std. (These are the z-score numbers.)
3. Work out the score for every usable bar in that pile (usable = every measurement that timeframe needs is present).
4. Line all those scores up smallest to biggest and read off the values at the 20%, 40%, 60% and 80% marks. These four cut-offs are named p20, p40, p60, p80. ("p80" = the score that 80% of all bars come in below, so only the top 20% beat it.)
5. Optional: also save the score at every 1% mark from 0 to 100, if you want a 0-to-100 number as well as a letter.

Bigger and broader is better here. You want a fair, stable picture of what "normal" looks like, so a single bar can be judged against it.

Step 4: turn the score into a letter

Compare the bar's score with the four cut-offs:

```
grade = A if score >= p80    (top 20%, most expanded)
        B if score >= p60    (next 20%)
        C if score >= p40    (middle)
        D if score >= p20    (below middle)
        E otherwise          (bottom 20%, most dormant)
```

A is the most expanded and awake (top fifth of all bars). E is the most dormant and asleep (bottom fifth).

If a bar does not have enough history yet to work out its measurements, give it NO grade at all (a blank), never an E. A blank means "don't know yet"; an E means "definitely asleep". Muddle the two and you poison your numbers.

If you saved the 0-to-100 marks in step 5, the 0-to-100 "maturity score" is just where this bar's score lands among them (95 means it is more expanded than 95% of all bars). Same idea as the letter, just finer.

A grade is a position, not a verdict

Important, and easy to get wrong: do NOT read A as "good" and E as "bad". These are not quality scores. They are positions in a cycle, the stage a stock is at in its wave of movement. A just means the stock is expanding hard right now; E means it is quiet. Quiet often means early, a stock about to wake up. And straight A's across all three timeframes usually means you are late, buying the top after the move has already run.

The edge is in how the three letters line up across daily, weekly and monthly, not in chasing the highest grade. To actually read them, the ripple from one timeframe to the next and which combinations carried an edge in the backtests, read the blog post and the combo-stats files that come with it: hoiboy.uk/posts/maturity-grading-from-backtest-data

Reference implementation (Python, pandas + numpy)

This is a clean, self-contained version of everything above. It is not my production code, it is the algorithm written out plainly so you (or your AI) can adapt it.

```
import numpy as np
import pandas as pd

ATR_PERIOD = 14
Z_CLIP = 4.0

# signed weights and per-timeframe windows
TF = {
    "daily":   {"weights": {"atr_pct": 1}, "coil": 30, "fast": 20},
    "weekly":  {"weights": {"atr_pct": 1}, "coil": 15, "fast": 10},
    "monthly": {"weights": {"atr_pct": 1, "breakout_hi": -1, "mom_atr": 1},
                "coil": 9, "fast": 10},
}


def wilder_atr(high, low, close, period=ATR_PERIOD):
    prev_close = close.shift()
    tr = pd.concat(
        [high - low, (high - prev_close).abs(), (low - prev_close).abs()],
        axis=1,
    ).max(axis=1)
    return tr.ewm(alpha=1.0 / period, min_periods=period).mean()


def compute_features(ohlcv, timeframe):
    """Per-bar causal features for one stock, one timeframe.
    ohlcv: DataFrame sorted oldest -> newest with 'high', 'low', 'close'.
    """
    h, low, c = ohlcv["high"], ohlcv["low"], ohlcv["close"]
    coil, fast = TF[timeframe]["coil"], TF[timeframe]["fast"]

    atr_pct = wilder_atr(h, low, c) / c
    breakout_hi = c / h.rolling(coil).max().shift(1) - 1.0
```

```
    mom_atr = ((c / c.shift(fast) - 1.0) / atr_pct.replace(0, np.nan)
               ).replace([np.inf, -np.inf], np.nan)

    return pd.DataFrame(
        {"atr_pct": atr_pct, "breakout_hi": breakout_hi, "mom_atr": mom_atr},
        index=ohlcv.index,
    )


def build_reference(feature_frames, timeframe):
    """Build the fixed reference from MANY stocks' feature frames (one tf).
    feature_frames: list of compute_features(ohlcv, timeframe) over your pool.
    Returns z-score params, grade breakpoints, and a 0..100 score grid.
    """
    weights = TF[timeframe]["weights"]
    pool = pd.concat(feature_frames, ignore_index=True)

    z_params = {}
    score = pd.Series(0.0, index=pool.index)
    gradeable = pd.Series(True, index=pool.index)
    for feat, sign in weights.items():
        mean, std = float(pool[feat].mean()), float(pool[feat].std())
        z_params[feat] = {"mean": mean, "std": std}
        z = ((pool[feat] - mean) / std).clip(-Z_CLIP, Z_CLIP)
        score = score + sign * z
        gradeable = gradeable & pool[feat].notna()

    score = score[gradeable]
    breakpoints = {f"p{p}": float(np.percentile(score, p)) for p in (20, 40, 60, 80)}
    grid = [float(x) for x in np.percentile(score, np.arange(101))]
    return {"weights": dict(weights), "z_params": z_params,
            "breakpoints": breakpoints, "grid": grid}


def grade_bar(feature_row, timeframe, reference):
    """Grade ONE bar's features. Returns (maturity_score 0..100, letter),
    or (None, None) when the bar has insufficient history."""
    weights = TF[timeframe]["weights"]
    score = 0.0
    for feat, sign in weights.items():
        val = feature_row.get(feat)
        if val is None or (isinstance(val, float) and np.isnan(val)):
            return (None, None)  # honest blank, never a fake E
        zp = reference["z_params"][feat]
        z = max(-Z_CLIP, min(Z_CLIP, (val - zp["mean"]) / zp["std"]))
        score += sign * z

    bp = reference["breakpoints"]
    if   score >= bp["p80"]: letter = "A"
    elif score >= bp["p60"]: letter = "B"
    elif score >= bp["p40"]: letter = "C"
    elif score >= bp["p20"]: letter = "D"
    else:                    letter = "E"

    pct = float(np.interp(score, reference["grid"], np.arange(101.0)))
    maturity_score = int(round(min(100.0, max(0.0, pct))))
    return (maturity_score, letter)


# --- usage sketch ---
# refs = {tf: build_reference([compute_features(df, tf) for df in many_stocks], tf)
#         for tf in ("daily", "weekly", "monthly")}
# feats = compute_features(one_stock_daily_ohlcv, "daily")
# print(grade_bar(feats.iloc[-1].to_dict(), "daily", refs["daily"]))   # latest bar
```

My actual calibration (optional)

If you build the reference yourself from your own stocks (which is the right move), you will get your own numbers. These are mine, from a large multi-year pool (roughly 6.2 million daily bars, 1.28 million weekly, 267 thousand monthly), in case you want to check your code produces something in the same ballpark.

```
DAILY
  atr_pct:     mean  0.043501,  std 0.051147
  breakpoints: p20 -0.40721,  p40 -0.24356,  p60 -0.04371,  p80 0.28982

WEEKLY
  atr_pct:     mean  0.104537,  std 0.107969
  breakpoints: p20 -0.47058,  p40 -0.29976,  p60 -0.09013,  p80 0.27306

MONTHLY
  atr_pct:     mean  0.310185,  std 2.837166
  breakout_hi: mean -0.209930,  std 0.250124
  mom_atr:     mean  1.814856,  std 7.587687
  breakpoints: p20 -0.68230,  p40 -0.38942,  p60 -0.04568,  p80 0.51794
```

How to use it in practice

1. Build the three references once, save them to a file, and reload them. Rebuild every few months as your data grows.
2. For each stock and timeframe, work out the measurements, then grade either the latest bar (a live read) or the bar at a past date (a backtest entry). The same code does both.
3. Read the three letters together, not one at a time. The blog post explains the pattern I look for: the daily firing while the bigger timeframe is still cold (room left to run), rather than everything maxed out at once (too late).
4. Treat it as a contributing nudge, not a magic filter. On my data it adds a modest lift, it does not pick trades on its own. It rides alongside the rest of my selection.

A few honest caveats

- The daily grade is the cleanest signal. Weekly and monthly are noisier on their own. They earn their keep as part of the three-timeframe read, not as standalone filters.
- Fit the reference to the stocks you actually trade. A reference built on large US stocks will not describe small-cap behaviour well.
- This grades expansion only. It says nothing about direction, fundamentals, or risk. It is one input among many.
- Garbage in, garbage out. Clean your price data first (splits, bad ticks, gaps).

Not advice, and not my problem if it goes wrong

None of this is financial advice, and I take no responsibility for what you do with it. It is here for you to learn from, adapt, and play around with. If you win, great. If you lose, that is on you, not me. It is not like I charged you a pretty penny for it. Have fun with it. Built with Claude Code.