

Ask me one question at a time so we can develop a thorough, step-by-step spec for this idea. Each question should build on my previous answers, and our end goal is to have a detailed specification I can hand off to a developer. Let's do this iteratively and dig into every relevant detail. Remember, only one question at a time.

Here's the idea:

Create a command-line tool that reads a CSV file of daily temperature readings, detects anomalies (days that differ by more than 2 standard deviations from a 30-day rolling average), and outputs:

A list of flagged dates

The magnitude of deviations

A simple ASCII chart highlighting anomalies

Constraints:

Must validate the schema

Must explicitly handle missing values

Must not silently drop rows

Must include tests

Must follow AGENTS.md guidelines

First question: what exact CSV schema do you want to require—**column names, data types, and date format** (e.g., date as YYYY-MM-DD and temperature as float), and do you want to allow any optional columns (and if so, which ones)?



Requires "Date" specified as YYYY-MM-DD and "Temperature" as a float. Other columns may exist, but they are not important.

Great — that gives us the base schema.

Next question:

**How should the tool behave if the required columns are missing or malformed?**

For example:

- If `Date` or `Temperature` is missing from the header
- If a row contains an invalid date (not `YYYY-MM-DD` )
- If `Temperature` cannot be parsed as a float

Should the tool:

1. **Fail fast** and stop with a clear error message, or
2. **Continue processing but report the problematic rows**, and if so, **how should those rows appear in the output report?**



Continue processing, but report the rows with data issues at the end of the output.

Got it.

Next question (missing values, explicitly): **What counts as a “missing Temperature” value in the CSV, and how should it be handled?**

Examples of “missing” you might want to treat as missing:

- empty cell
- whitespace-only
- `NA` , `N/A` , `null` , `NULL` , `NaN`

And handling options (since we must not silently drop rows):

- keep the row and mark it as “invalid/missing” (excluded from calculations)
- treat missing temperature as zero (usually a bad idea, but explicit)

- fail anomaly detection for that date but still list it in the issues report

Which set of missing tokens should we recognize, and what should we do with those rows for rolling stats/anomaly detection?



Any value that is not a number should be treated as missing. these should be reported as errors.

Understood.

Next question: **How should missing/invalid temperature rows affect the 30-day rolling window calculation?**

For example, if a day inside the rolling window has a missing/invalid temperature:

1. **Exclude it from the rolling calculation** (window may contain fewer than 30 valid values).
2. **Require 30 valid values**; if any are missing, the anomaly check for that day cannot be computed.
3. **Invalidate the entire window** and report that the anomaly check was skipped for that date.

Which behavior should the tool implement?



Exclude it from the rolling calculation

Cool.

Next question: **What's the minimum number of valid prior days required to compute the rolling average/stddev and perform anomaly detection for a date?**

Examples:

- require at least **30** valid prior temps (strict)
- require at least **N** valid prior temps (e.g., 20)
- allow any count  $\geq 2$  (since stddev needs at least 2), but results may be noisy

What threshold do you want?



let's require at least 10 valid prior temperatures

Good choice — that keeps early data usable while still giving a meaningful standard deviation.

Next question:

**Should the 30-day rolling window be defined by *calendar days* or by the *previous 30 valid rows in the file*?**

Two possible interpretations:

### 1. Calendar-based window

- Look at readings from the **previous 30 calendar days** before the current date.
- If some days are missing entirely from the dataset, they simply don't contribute values.

### 2. Row-based window

- Use the **previous 30 rows in the CSV** regardless of their actual date spacing.

Calendar-based is usually more correct for time-series, but row-based is simpler and sometimes used for sensor logs.

Which behavior should the tool implement?



use the previous 30 valid rows in the file

Understood.

Next question:

**Should the CSV be required to be sorted by Date , or should the tool handle unsorted input?**

Possible behaviors:

1. **Require sorted input** — validate that dates are strictly increasing and report errors if not.
2. **Automatically sort by Date before processing** (while still reporting any parsing issues).
3. **Process rows exactly in file order**, ignoring date ordering entirely (treating the CSV as a sequential log).

Which approach do you want?



Require sorted input. Stop the program if dates are found out of order and report that error condition.

Got it.

Next question: **What should "sorted" mean exactly—are duplicate dates allowed?**

Options:

1. **Strictly increasing** dates only (no duplicates).
2. **Non-decreasing** allowed (duplicates OK), and duplicates are treated as separate readings.
3. Duplicates allowed, but **treated as an error** (stop like out-of-order).

Which do you want?



Dates must be strictly increasing

Perfect.

Next question: **Anomaly rule detail — when you say “differ by more than 2 standard deviations from a 30-day rolling average,” do you mean:**

- Flag if  $\text{abs}(\text{temp} - \text{rolling\_mean}) > 2 * \text{rolling\_stddev}$   
(two-sided: both unusually hot and unusually cold)

or

- Only flag unusually high temps (  $\text{temp} - \text{mean} > 2 * \text{stddev}$  ), or only unusually low?

And confirm: use **strictly greater than** (  $> 2 * \text{sd}$  ), not  $\geq$  ?



two sided - both unusually hot and unusually cold

Great.

Next question: **How should the magnitude of the deviation be reported for flagged anomalies?**

Common options:

**1. Raw temperature difference**

$\text{temp} - \text{rolling\_mean}$  (e.g.,  $+5.3^\circ\text{F}$  or  $-4.8^\circ\text{F}$ )

**2. Absolute difference**

$\text{abs}(\text{temp} - \text{rolling\_mean})$  (always positive)

**3. Z-score**

$(\text{temp} - \text{rolling\_mean}) / \text{rolling\_stddev}$

**4. Both raw difference and Z-score**

Which format should appear in the anomaly output?



both the raw difference and the z-score

Good — that will make the output more useful for analysis.

Next question: **What temperature units should the tool assume (or display)?**

Options:

1. **Assume Fahrenheit** (no conversion, just label output as °F)
2. **Assume Celsius**
3. **Unit-agnostic** (just treat values as numbers and don't label units)
4. **Allow a CLI flag** like `--units F|C` that only affects labeling

Which behavior do you want?



Assume fahrenheit

Next question:

**How should the ASCII chart be structured and scaled?**

There are several design choices we need to lock down so a developer can implement it consistently:

### 1. Chart orientation

- One **row per day** (timeline vertically) with a marker showing temperature position on a horizontal scale
- Or a **traditional plot** with temperature on the vertical axis and time on the horizontal axis

### 2. Scale

- Fixed width (e.g., 60 characters)

- Automatically scaled to the **min/max temperature in the dataset**
- Or a **fixed temperature range** (e.g., 0–120°F)

### 3. Anomaly highlighting

- Replace the normal marker with something like `!` or `*`
- Color (if terminal supports it)
- Both

What chart style do you want?



let's use a one row per day (vertical timeline). A fixed width of 70 characters that is scaled to the min/max in the dataset. Highlight anomalies with a '#' sign

Nice—clear and implementable.

Next question: **What should each chart line include, exactly?**

For example, something like:

- `YYYY-MM-DD |<70-char bar with marker>| temp=72.3 mean=... z=...`  
or a more compact:
- `YYYY-MM-DD 72.3F |-----#-----|`

Which fields do you want printed on every line (date, temp, marker-only, mean, z-score, etc.), and do you want anomaly lines to also include the deviation numbers inline or only in the anomaly list section?



use the compact form

Got it.

Next question: in that compact per-day line, **do you want to include the temperature value as a number, or should the line be just the date plus the 70-char chart?**

Examples:

A) 2026-03-01 72.3F |-----#-----...|

B) 2026-03-01 |-----#-----...|

Which one?



The date, the position of the line, and then a label align on the right with the actual temperature

Great.

Next question: **What character should represent a normal (non-anomalous) temperature point on the chart?**

Right now we have:

- # = anomaly

For normal days we need a marker. Common options would be:

- \*
- |
- o
- .

Example line:

```
2026-03-01 |-----o-----| 72.3F
2026-03-02 |-----#-----| 89.1F
```



Which character should represent **normal temperature points**?



use an '\*'

Next question:

**What should the command-line interface look like?**

Specifically:

**1. How is the input CSV provided?**

- Positional argument (e.g., `temp-anomaly data.csv` )
- Named flag (e.g., `temp-anomaly --input data.csv` )
- Allow reading from **stdin** as well?

**2. Where should the output go?**

- Always **stdout**
- Optional `--output file`
- Separate outputs for **chart**, **anomaly list**, and **error report**?

What CLI behavior do you want?



positional argument and then always to stdout

Next question:

**What programming language and runtime should the tool target?**

This affects packaging, testing framework, dependency policy, and how the CLI is implemented.

Common options might be:

- **Python** (e.g., Python 3.11+, `argparse` , `pytest` )
- **Go** (single static binary, `go test` )
- **Rust** ( `clap` , `cargo test` )

- **Java** (since you often work in that ecosystem)

Which language should the developer implement this tool in?



use Python, with pytest. Since this is simple, just use `sys.argv` to get the filename

Next question:

**Are third-party Python dependencies allowed, or should it be standard library only (besides pytest for tests)?**

This matters for CSV parsing and rolling stats:

- Stdlib-only: `csv`, `datetime`, `statistics` (but rolling/stddev handling needs careful implementation)
- Allow `pandas` / `numpy` for simpler rolling calculations (heavier deps)

Which do you want?



you can use third party dependences. Create a `requirements.txt` file. Use a virtual environment called `"venv"`. Just use `pip` for the installs

Next question:

**How should the output be structured on stdout?**

For example, do you want clearly separated sections like this:

```
TEMPERATURE ANOMALY REPORT
=====
```



## ASCII CHART

-----

<chart lines here>

## ANOMALIES

-----

Date	Temp(F)	Mean(F)	Diff(F)	Z-Score
2025-02-14	89.2	72.1	+17.1	2.41

...

## DATA ISSUES

-----

Row 17: Temperature missing or not numeric

Row 42: Temperature missing or not numeric

Or do you want a **different ordering or formatting** of these sections?



Your sample looks great!

Next question:

**How should row numbers be defined in the "DATA ISSUES" section?**

Possible interpretations:

### 1. CSV line number including header

- Header = line 1
- First data row = line 2

### 2. Data row index only

- First data row = row 1

### 3. Both (e.g., Line 17 (data row 16) )

This matters for debugging corrupted files. Which format should the tool report?



Next question:

**How should the tool behave if the rolling standard deviation is zero?**

This can happen if the previous temperatures are all identical. In that case the Z-score formula would cause a divide-by-zero.

Reasonable options:

1. **Do not flag anomalies for that row** and report  $z\text{-score} = \text{N/A}$
2. **Treat any deviation from the mean as an anomaly**
3. **Skip anomaly evaluation for that row** and report it in the data issues section
4. **Fail the program**

Which behavior should the tool implement?



Next question:

**Should the anomaly detection use the rolling window that includes the current temperature, or only the prior temperatures?**

Two possible definitions:

1. **Previous rows only** (most common in anomaly detection)
  - Rolling stats computed from the **previous  $\leq 30$  valid rows**, excluding the current row.
2. **Include the current row**
  - Rolling stats computed from the window **including the current temperature**.

Option 1 avoids the anomaly influencing its own statistics.

Which behavior should the tool use?



1

Next question:

**Should the tool enforce that the Date column appears exactly as "Date" and Temperature exactly as "Temperature" (case-sensitive), or should it accept variations like date , DATE , Temp , etc.?**

Options:

- 1. Strict schema** – must be exactly Date and Temperature .
- 2. Case-insensitive** – accept date , DATE , etc., but still require those names semantically.
- 3. Flexible aliases** – allow things like Temp , temperature\_f , etc.

Given your earlier constraint about **schema validation**, which level of strictness do you want?



2

Next question:

**How should the tool behave if the CSV contains fewer than 10 valid temperatures total?**

Recall we require **at least 10 prior valid temperatures** to perform anomaly detection.

Possible behaviors:

1. **Run normally but produce no anomalies**, since the threshold is never met.
2. **Print a warning section** like "INSUFFICIENT DATA FOR ANOMALY DETECTION."
3. **Treat it as an error and exit with a non-zero status code.**

Which behavior should the tool implement?



1

Next question:

**What exit codes should the program use?**

A simple and useful convention might be:

- **0** — program ran successfully (even if anomalies or data issues were found)
- **1** — fatal input error (e.g., file not found, CSV unreadable, dates not strictly increasing)
- **2** — schema error (required columns missing)

Or we could simplify to just **0 for success / 1 for failure**.

What exit code policy do you want?



Let's follow your convention

Next question:

**Where should the AGENTS.md guidelines apply, and what specific rules must the developer follow from it?**

Since you explicitly required "*must follow AGENTS.md guidelines*", we need to clarify what that means in the spec. For example, AGENTS.md files often include rules like:

- repository structure conventions
- code documentation expectations
- commit message format
- testing requirements
- linting/formatting tools (e.g., `black` , `ruff` , `flake8` )
- instructions for how agents/tools should run tests

So for this project:

1. **Will there be an AGENTS.md file in the repo that the developer must follow?**
2. If so, **what specific requirements should it enforce** (linting, docstrings, test coverage, CLI help text, etc.)?

We should lock those down so the spec is complete.



Yes. We'll need to create an AGENTS.md as part of the creation process.

Next question: **What should AGENTS.md require, specifically?**

Pick the rules you want enforced (we'll write them into AGENTS.md verbatim). Common choices:

- Code formatting: `black`
- Linting: `ruff` (or `flake8` )
- Type checking: `mypy`
- Testing: `pytest` with a minimum coverage threshold (e.g., `--cov ≥ 90%`)
- Pre-commit hooks required ( `pre-commit` )
- Project layout (e.g., `src/` + `tests/` )
- Docstrings required for public functions
- "No pandas" / "pandas allowed" rule
- "All user-facing errors must be actionable" rule
- "Do not print stack traces on expected user errors"

Which of these (or others) do you want to mandate?



Let's go with those. Plus look at my earlier answers in this conversation

Next question: **Do you want this packaged as an installable CLI command (e.g., `temp-anomaly` via `pip install -e .` with a console script entry point), or is running it as `python temp_anomaly.py <file.csv>` sufficient?**



just running it as `python temp_anomal.py <file.csv>` is sufficient

Next question:

**What Python version should the project target and require?**

This should be specified in the spec, `AGENTS.md`, and `requirements.txt` so developers know what language features and tooling are allowed.

Typical options:

- **Python 3.10+**
- **Python 3.11+**
- **Python 3.12+**

Which minimum version should the project require?



Next question:

### What should the project directory structure look like?

For example, a simple but disciplined layout might be:

```
temp-anomaly/
├── temp_anomaly.py
├── requirements.txt
├── AGENTS.md
├── README.md
├── tests/
│   ├── test_schema_validation.py
│   ├── test_anomaly_detection.py
│   ├── test_missing_values.py
│   └── test_chart_output.py
└── sample_data/
    └── example.csv
```



Or we could enforce a stricter layout (e.g., `src/` structure).

Do you want:

1. **Single-script layout** (like above, simpler), or
2. `src/` **project layout** with modules?



Next question:

**Should the tool support large CSV files (e.g., millions of rows), and if so, should it process the file in a streaming/iterative way instead of loading the entire dataset into memory?**

Two reasonable options:

1. **Simple approach** – read the entire CSV into memory (simpler implementation).
2. **Streaming approach** – process rows incrementally while maintaining the rolling window (more scalable and memory-efficient).

Which approach should the specification require?



1

Next question:

**For standard deviation, do you want the rolling stddev computed as population ( `ddof=0` ) or sample ( `ddof=1` )?**

This affects the threshold and thus which days get flagged.

- **Population stddev (`ddof=0`)** is common for descriptive stats over a window.
- **Sample stddev (`ddof=1`)** is common for estimating variance from a sample.

Which one should we use?



Use the sample

Next question:

**How should the tool behave if the input file does not exist or cannot be opened?**

Options:

1. Print a clear error like  
`ERROR: Cannot open file '<filename>'`  
then exit with **exit code 1**.
2. Print the full Python exception message.

3. Print a short message plus the exception reason.

What exact behavior do you want?



1

Next question:

Should the tool detect and report duplicate headers or multiple Date / Temperature columns (case-insensitive), or simply use the first matching column it finds?

Possible behaviors:

1. Treat duplicates as a schema error and exit with code 2.
2. Use the first occurrence and ignore the rest.
3. Warn but continue.

Given your emphasis on **explicit schema validation**, which rule should we enforce?



1

Next question:

How should the tool handle rows where the Date value is syntactically valid ( YYYY-MM-DD ) but represents an impossible date (e.g., 2025-02-30 )?

Possible behaviors:

1. Treat it as a **data issue** (like missing temperature): report the CSV line number in **DATA ISSUES** and continue processing.
2. Treat it as a **fatal error** and stop the program.

3. Skip the row silently (not allowed by your earlier constraint, but listing for completeness).

Which behavior should the tool implement?



1

Next question:

**Should rows with data issues (invalid date or non-numeric temperature) still appear in the ASCII chart timeline, or should the chart only include valid rows?**

Possible behaviors:

1. **Exclude them from the chart** entirely but list them in **DATA ISSUES**.
2. **Include them in the chart** with a placeholder marker (e.g., ? ) and still report them as issues.
3. **Include the date but no marker.**

This affects how faithfully the chart reflects the raw file vs. the analyzed dataset. Which behavior do you want?



1

Next question:

**Do you want the anomaly list to be sorted by date (same as input order) or sorted by "most extreme" first (highest absolute z-score)?**

Options:

1. **Date order** (chronological, same as file)

## 2. Severity order (descending $\text{abs}(z)$ ), with date as a tiebreaker

Which should it be?



1

Next question:

**Should the rolling mean and standard deviation be computed using only the valid temperatures that appear before the current row, but still limited to the most recent 30 valid rows?**

To confirm the intended behavior precisely:

- Maintain a window of **up to the previous 30 valid temperatures**
- Rows with invalid/missing temperature **do not enter the window**
- The window **slides forward only when a valid temperature appears**

Example:

```
valid temps: 70, 71, 72, [invalid], 73  
window for 73 → [70,71,72]
```



Is this exactly the behavior you want for maintaining the rolling window?



Yes. that is correct

Great. Next question:

**Should the tool produce a helpful usage message if the user runs it incorrectly (e.g., no arguments or too many arguments)?**

Example behavior:

```
Usage: python temp_anomaly.py <input.csv>
```



Then exit with **code 1**.

Do you want exactly this behavior, or should it include additional help text (for example explaining the required schema)?



of course

Next question:

**Should the tool support different CSV delimiters (like tab-separated), or require standard comma-separated CSV only?**

Options:

1. Require comma-separated only.
2. Auto-detect delimiter.
3. Add an optional flag (but you prefer `sys.argv` minimalism, so likely not).

Which do you want?



1

Next question:

**Should the ASCII chart show the entire dataset, or should it truncate very large datasets?**

For example, if the CSV has **50,000 rows**, printing a full chart could flood the terminal.

Possible policies:

1. **Always print the full chart** regardless of size.
2. **Cap the chart** (e.g., first 500 rows) and report truncation.
3. **Only chart rows with valid temperatures** (still potentially very large).

Which behavior should the specification require?




1

Excellent — I think we now have nearly all behavioral details locked down.


**Next question (last major behavioral gap):**

Should the ASCII chart show a **baseline axis or boundary markers** for the 70-character scale, or just the plotted line?

Example with boundaries:

```
2025-01-01 |-----*-----| 72.3F 
```

Example without boundaries:

```
2025-01-01 -----*----- 72.3F 
```

The boundary bars ( | ) make the scale visually clearer but are slightly noisier. Which format do you want?



use the boundary bars and identify those values at the bottom of the chart

Great. Next question:

### How should the min/max scale labels appear at the bottom of the chart?

Since the chart width is **70 characters**, we need a clear way to label the boundaries.

Possible formats:

#### Option A — simple boundary labels



#### Option B — aligned with the bars



#### Option C — labeled axis



Which format do you want for the bottom scale labels?



B

Next question:

### How should temperatures be formatted when printed?

We need to lock the numeric precision for consistent output and tests.

Options:

1. **1 decimal place** (e.g., 72.3F )
2. **2 decimal places** (e.g., 72.34F )
3. **Preserve original precision from the CSV**

This affects:

- chart labels
- anomaly table values
- min/max scale labels
- mean and diff values in the anomaly report

Which precision should the tool use?



1

Next question:

**How should the rolling mean and standard deviation be rounded/formatted in the anomaly table?**

Since temps are 1 decimal, we can either:

1. Also print **mean and stddev to 1 decimal**, and z-score to (say) 2 decimals
2. Print mean/stddev to **2 decimals** for more accuracy, z-score to 2 decimals
3. Mean 1 decimal, stddev 2 decimals, z-score 2 decimals (mixed)

What do you want for:

- mean (°F)
- diff (°F)
- z-score (unitless)

(We already agreed temp prints with 1 decimal.)



Next question:

**Should the anomaly report include the rolling mean used for the comparison, or only the temperature, difference, and z-score?**

Two options:

**Option A — include mean (more transparent)**

Date	Temp(F)	Mean(F)	Diff(F)	Z-Score
2025-07-14	95.1	72.4	+22.7	2.41



**Option B — omit mean (simpler)**

Date	Temp(F)	Diff(F)	Z-Score
2025-07-14	95.1	+22.7	2.41



Which format should the specification require?



A

Next question:

**Should the anomaly report explicitly indicate whether the anomaly is "HIGH" or "LOW", or should that be inferred from the sign of  $\text{Diff}(F)$  / the z-score?**

Two options:

**Option A — explicit direction column**

Date	Temp(F)	Mean(F)	Diff(F)	Z-Score	Type
2025-07-14	95.1	72.4	+22.7	2.41	HIGH
2025-12-02	31.0	54.6	-23.6	-2.18	LOW



**Option B — no extra column**, direction implied by sign of Diff(F) and Z-Score .

Which should the tool use?



B

Next question:

**Should the tool include a summary section reporting counts**, such as:

- total rows read
- valid temperature rows
- rows with data issues
- number of anomalies detected

Example:

SUMMARY

-----

Total rows read: 365

Valid temperature rows: 360

Data issues: 5

Anomalies detected: 7



Or should the output contain **only the chart, anomaly list, and data issues sections** without a summary?



No

Next question:

### Should the tool verify that the CSV file actually contains a header row?

Right now the spec assumes a header with `Date` and `Temperature`. But some CSV files begin directly with data.

Possible behaviors:

1. **Require a header row** and treat absence as a **schema error (exit code 2)**.
2. Attempt to **infer columns** if the header is missing.
3. Treat the first row as data and fail later when required columns are missing.

Given your emphasis on **explicit schema validation**, which behavior should the specification require?



Yes. Must require a header row

Next question:

### Should the program tolerate leading/trailing whitespace in header names and values?

Examples:

```
Date , Temperature
2025-01-01 , 72.3
```



Possible behaviors:

1. **Trim whitespace** around headers and values before validation (more robust).
2. Treat whitespace as **schema/data errors** (strict parsing).

Which rule should the specification enforce?



yes

Next question:

**Should the tool support comment lines in the CSV (e.g., lines starting with # ) or treat every non-header line as data?**

Possible behaviors:

1. **No comments allowed** — every line after the header must be a CSV row.
2. **Allow comment lines starting with #** and ignore them.
3. Allow **blank lines only**, but no comments.

Which behavior should the specification require?



1

Next question:

**Should the CSV parser enforce that each data row has at least as many columns as the header, or should short rows be treated as data issues?**

Example problematic row:

```
Date, Temperature, Station  
2025-01-01, 72.1, KXYZ  
2025-01-02
```



Possible behaviors:

1. **Treat short/malformed rows as data issues** and report the CSV line number.
2. **Treat malformed rows as fatal input errors** and exit.
3. **Attempt to pad missing columns** (not recommended).

Which rule should the tool follow?



1

Excellent — we're very close to having a complete specification.

Next question:

**Should the tool verify that the `Temperature` column does not contain extra characters like units (e.g., `72.3F` , `68°F` ), or should it attempt to parse the numeric portion?**

Possible behaviors:

- 1. Strict numeric parsing only** — values like `72.3F` are treated as invalid and reported in **DATA ISSUES**.
- 2. Lenient parsing** — extract the numeric portion if possible.

Given your earlier rule *"any value that is not a number should be treated as missing and reported as an error"*, I suspect the answer is **1**, but I want to confirm explicitly for the spec.



1

Next question:

**Do you want the tool to treat `Temperature` values like `inf` , `-inf` , or `nan` (which Python can parse as floats) as valid numbers, or should those be treated as invalid/missing and reported as data issues?**

Options:

1. Treat `inf`, `-inf`, `nan` as **invalid/missing** (reported as issues).
2. Treat them as **valid floats** (and they influence stats).

Which behavior should the spec require?



1

Next question:

**Should the anomaly table print a + sign for positive differences, or only show the minus sign for negative values?**

Examples:

With explicit sign:

```
+5.3  
-4.8
```



Without explicit positive sign:

```
5.3  
-4.8
```



Explicit signs often make anomaly direction clearer in tabular reports.

Which format should the tool use?



use explicit signs

Next question:

Should the tool validate that the CSV file is UTF-8 encoded, or simply rely on Python's default file reading behavior and fail if decoding errors occur?

Options:

1. **Explicitly open with UTF-8 and fail if decoding fails** (clearer and reproducible).
2. **Use Python defaults** and let decoding errors propagate.

Which behavior should the spec require?



2

Next question:

Should the tool validate that the `Date` column values are unique, or only enforce that they are strictly increasing?

Right now the spec says **strictly increasing**, which implicitly prevents duplicates, but we could also explicitly validate uniqueness for clearer error reporting.

Options:

1. **Only enforce strictly increasing order** (duplicates automatically fail that check).
2. **Explicitly check for duplicates and report a specific error message.**

Which approach do you want in the specification?



2

Next question:

Should the tool require that the rolling window uses exactly the last 30 valid temperatures when available, or allow the window to grow gradually until it reaches

30?

Example behavior:

- First valid row → window size = 0
- Row 2 → window size = 1
- ...
- Row 10 → first possible anomaly check (since you required  $\geq 10$ )
- Row 31 → window capped at 30

This is the typical **progressively growing window capped at 30** behavior, but I want to confirm that's exactly what you want.



the window can gradually grow

Next question:

**Should the anomaly threshold (currently “more than 2 standard deviations”) be hard-coded, or should it be defined as a constant in the code so it can be easily changed later?**

Options:

1. **Hard-coded literal** 2 in the logic.
2. **Named constant** like `ANOMALY_STDDEV_THRESHOLD = 2` .

Option 2 usually improves maintainability and clarity.

Which should the spec require?



hard coded as a named constant

Next question:

**Should the project include a `README.md` , and if so, what sections should it contain?**

A typical minimal README for this tool might include:

- **Overview** (what the tool does)
- **Requirements** (Python 3.12+, venv, requirements.txt)
- **Setup instructions** ( `python -m venv venv , pip install -r requirements.txt` )
- **Usage** ( `python temp_anomaly.py <input.csv>` )
- **Required CSV schema**
- **Example input**
- **Example output**
- **Running tests** ( `pytest` )

Do you want all of those sections included, or a smaller README?



Yes. Include all of that

Great. **Final specification question:**

**Should the tests include fixed “golden output” comparisons for the ASCII chart and report sections, or should they test components separately (e.g., anomaly detection logic, chart generation, schema validation) without requiring an exact full-output match?**

Options:

1. **Golden file tests** — compare the program’s full stdout against an expected output file (very strict, good for CLI tools).
2. **Component tests** — test logic pieces independently and only partially check output.
3. **Both** — unit tests for logic plus at least one end-to-end golden output test.

Which approach should the specification require?



Perfect — that completes the remaining key decisions.

If you'd like, next I can **assemble the full handoff spec** (requirements, behavior, edge cases, output format, project structure, AGENTS.md contents, and test plan) in one clean document.



Now that we've wrapped up the brainstorming process, can you compile our findings into a comprehensive, developer-ready specification?

Include all relevant requirements, architecture choices, data handling details, error handling strategies, and a testing plan so a developer can immediately begin implementation.

## Developer-ready specification: `temp_anomaly.py` CSV temperature anomaly detector

### 1. Goal

Build a command-line tool that:

- Reads a CSV file containing daily temperature readings.
- Detects anomalies where a day's temperature differs from a rolling mean by **more than 2 standard deviations**, using a **rolling window of the previous 30 valid rows**.
- Outputs to **stdout**:
  1. An ASCII chart (vertical timeline) highlighting anomalies.
  2. A chronological table of flagged anomaly dates with deviation magnitude.
  3. A chronological report of data issues (invalid rows).

## 2. Implementation constraints and choices

- **Language:** Python 3.12+
- **Invocation:** `python temp_anomal.py <input.csv>`
  - **Input filename** is a **single positional argument** read from `sys.argv`.
  - Output always to **stdout**.
- **Dependencies:**
  - Third-party dependencies allowed; provide `requirements.txt`.
  - Use a virtual environment named `venv` (documented in README).
- **Stats behavior:**
  - Rolling window uses **previous 30 valid temperature rows in the file** (row-based, not calendar-based).
  - Rolling stats computed from **prior rows only** (exclude current row).
  - Standard deviation uses **sample** standard deviation (**ddof = 1**).
  - Minimum requirement: **at least 10 valid prior temperatures** required to compute mean/stddev and evaluate anomaly.
  - If rolling stddev is **0**, do **not** flag anomalies for that row and report **z-score as N/A** for the anomaly logic context (but see output requirements below—only anomalies are printed in the anomaly table).
- **Schema strictness:**
  - Required columns: `Date` and `Temperature`.
  - Header matching is **case-insensitive** (e.g., `date`, `DATE` accepted).
  - Leading/trailing whitespace should be **trimmed** from headers and values before validation/parsing.
  - If multiple columns match `Date` or `Temperature` (case-insensitive), treat as **schema error** and exit.
  - Must require a **header row**.
- **Units:** Assume **Fahrenheit**; label output as `F`.

## 3. Expected CSV schema and parsing rules

### Required columns

- **Date:** `YYYY-MM-DD`
- **Temperature:** `float`

Other columns may exist and must be ignored.

### Row validity rules

A row is considered to have a **data issue** (non-fatal) if any of the following occurs:

- Date value is not parseable as a real calendar date in YYYY-MM-DD format (including impossible dates like 2025-02-30).
- Temperature is not strictly numeric (no unit suffixes like 72.3F), after trimming whitespace.
- Temperature parses to a float but is **NaN** or **±inf** → treat as invalid/missing.
- The row is malformed/short relative to the header (e.g., missing columns) → treat as a data issue.

#### Missing temperature definition:

- "Any value that is not a number" counts as missing/invalid.
- Invalid/missing temperature rows must be **reported as errors** (data issues) and **excluded from rolling calculations**.

#### Do not silently drop rows

- Rows with data issues must be recorded and reported at the end in a **DATA ISSUES** section.
- Rows with data issues must **not** appear in the chart and must not affect rolling stats.

#### 4. Input ordering constraints

- Input must be sorted by date in **strictly increasing** order.
- **Duplicates are not allowed**, and must trigger a specific error message (distinct from "out of order").
- If dates are out of order or duplicates are found:
  - Stop immediately (fatal).
  - Print a clear error message.
  - Exit with code **1**.
- Sorting must *not* be performed automatically.

#### 5. Anomaly detection specification

##### Rolling window definition

Maintain a rolling list of the **previous up to 30 valid temperature values** encountered *before* the current valid temperature row.

- Invalid temperature rows do not enter the window.
- The window grows gradually until it reaches 30.
- For a given row, compute stats from the window **excluding** current row.

## Eligibility to evaluate anomaly

- Only evaluate anomalies for rows with valid temperature **and** at least **10** valid prior temperatures in the rolling window.
- If fewer than 10 valid priors exist, the row is valid but simply not eligible for anomaly detection.

## Stats and threshold

- `mean` = average of rolling window
- `stddev` = sample standard deviation of rolling window ( `ddof=1` )
- If `stddev == 0` , do not flag anomalies; treat z-score as N/A for evaluation.
- An anomaly is flagged if:
  - `abs(temp - mean) > ANOMALY_STDDEV_THRESHOLD * stddev`
  - where `ANOMALY_STDDEV_THRESHOLD` is a **named constant** with hard-coded value `2` .
- Report:
  - `diff = temp - mean` (signed)
  - `z = diff / stddev` (signed), if `stddev > 0`

## 6. Output requirements (stdout)

Output must include sections in this order:

1. **TEMPERATURE ANOMALY REPORT** header
2. **ASCII CHART**
3. **ANOMALIES**
4. **DATA ISSUES**

No additional "summary" section.

### 6.1 ASCII chart

- Vertical timeline: **one line per valid row** (valid Date + numeric temperature only).
- Chart line format (compact):
  - Left: date
  - Middle: boundary bars and 70-char scale with marker
  - Right: temperature aligned as a trailing label

Example (illustrative):

ASCII CHART

-----



```

2026-03-01 |-----*-----| 72.
2026-03-02 |-----#-----| 89.
...
|-----|
12.4F                                           97.8F

```

#### Rules:

- Width of plotting region: **70 characters**
- Include **boundary bars**: | at both ends on each chart line.
- Normal points use marker \*
- Anomalies use marker #
- Scaling:
  - Use min/max temperature **across all valid rows** (not including invalid rows).
  - Map each temperature to an integer position within [0, 69] (70 slots).
  - If `min == max`, place marker at center or position 0 (implementation-defined, but must be deterministic and tested).
- Bottom axis:
  - After all chart rows, print:
    - A boundary line: | + 70 dashes + |
    - Then a line with **min value at far left and max value at far right** (Option B):
      - `min` and `max` formatted with **1 decimal** and F (no degree symbol required).
      - Align max label to the right edge under the final boundary.

#### 6.2 Anomalies table

- Chronological order (same as input order).
- Must include columns:
  - Date
  - Temp(F)
  - Mean(F)
  - Diff(F) (signed, include + for positive)
  - Z-Score (signed)
- Formatting:
  - Temperature, mean, diff: **1 decimal**
  - Z-score: **1 decimal**
  - Use explicit + for positive diff (and optionally z-score, but at least diff must have explicit sign; keep consistent).

- Only include rows that were eligible and flagged as anomalies.
- If there are **no anomalies**, show the header and either no rows or a clear “(none)” line (developer choice; must be consistent and tested).

### 6.3 Data issues section

- Listed at the end.
- Must include CSV **line numbers including header as line 1**.
- Must not stop execution for these issues; continue processing.
- Each issue should state:
  - Line number
  - A concise description (e.g., invalid date, non-numeric temperature, NaN/inf, malformed row)
  - Preferably include the raw value encountered (optional but helpful)

## 7. Error handling and exit codes

### Fatal errors (must terminate)

1. File cannot be opened:
  - Print: ERROR: Cannot open file '<filename>'
  - Exit code 1
2. Incorrect invocation (no args, too many args):
  - Print usage line: Usage: python temp\_anomal.py <input.csv>
  - Exit code 1
3. CSV schema errors:
  - Missing required columns ( Date / Temperature , case-insensitive match)
  - Duplicate matching columns (case-insensitive)
  - Missing header row
  - Print clear error message (actionable)
  - Exit code 2
4. Dates out of order or duplicates:
  - Stop immediately
  - Print clear error message specifying whether:
    - duplicate date found, or
    - out-of-order date found
  - Exit code 1

### Non-fatal errors (must continue)

- Invalid/missing temperature
- Invalid date (even if syntactically correct but impossible)
- Malformed row/short row
- These must be recorded and shown in **DATA ISSUES**.

## 8. Project structure

Single-script layout:

```
temp-anomaly/
├── temp_anomal.py
├── requirements.txt
├── AGENTS.md
├── README.md
└── tests/
    ├── test_schema_validation.py
    ├── test_anomaly_detection.py
    ├── test_missing_values.py
    └── test_chart_output.py
```

(Exact test filenames may vary, but must cover all behaviors.)

## 9. AGENTS.md requirements (must be created)

AGENTS.md must codify:

- Formatting: **black**
- Linting: **ruff**
- Type checking: **mypy**
- Testing: **pytest** with **coverage threshold** (e.g.,  $\geq 90\%$ ) using `pytest-cov`
- Pre-commit hooks: **pre-commit** required
- Layout: single-script layout as specified (no `src/`)
- Docstrings required for public functions
- Dependency rule: third-party deps allowed via `requirements.txt`
- Error handling rule: "All user-facing errors must be actionable" and "Do not print stack traces for expected user errors"

Also include a short "How to run" section consistent with README:

- create venv named `venv`
- `pip install -r requirements.txt`
- `pytest`

- formatting/lint/typecheck commands
- pre-commit install/run

## 10. README.md requirements

README must include:

- Overview
- Requirements (Python 3.12+, venv name `venv` )
- Setup instructions:
  - `python -m venv venv`
  - `activate venv`
  - `pip install -r requirements.txt`
- Usage:
  - `python temp_anomal.py <input.csv>`
- Required CSV schema (case-insensitive headers allowed; whitespace trimmed)
- Example input CSV
- Example output (sample chart + anomaly + data issues format)
- Running tests ( `pytest` , and coverage expectation)

## 11. Testing plan (pytest)

Testing must include **both** unit/component tests and end-to-end "golden output" tests.

### 11.1 Unit/component tests

Cover at least:

- Schema validation:
  - missing Date/Temperature → exit code 2
  - duplicate matching columns (case-insensitive) → exit code 2
  - header trimming and case-insensitivity
- Date ordering validation:
  - strictly increasing enforced
  - duplicates produce specific error and exit code 1
  - out-of-order produces specific error and exit code 1
- Missing/invalid values:
  - non-numeric temperature handled as data issue
  - `nan` , `inf` , `-inf` treated as data issue
  - invalid/impossible dates treated as data issue

- malformed/short rows treated as data issue
- ensure invalid rows are excluded from rolling window and chart
- Rolling window mechanics:
  - window uses previous 30 valid rows only
  - window grows gradually, anomaly checks begin only at  $\geq 10$  valid priors
  - sample stddev ( `ddof=1` ) behavior
  - `stddev == 0` behavior (no anomaly)
- Detection logic:
  - two-sided anomalies (high and low)
  - threshold strictly “more than” 2 stddev (not  $\geq$ )

## 11.2 Golden output tests (end-to-end)

At least one test that:

- Creates a small CSV fixture with:
  - valid rows
  - at least one high anomaly and one low anomaly
  - at least one invalid temperature
  - at least one invalid date
- Runs the script (e.g., via `subprocess.run` ) and asserts:
  - exit code is 0
  - stdout matches an expected “golden” output file/string exactly
  - includes chart markers `*` and `#` , correct min/max labels, and correct anomaly table formatting

Additional golden tests recommended:

- No anomalies case
- Only  $< 10$  valid temperatures case (no anomalies, still chart prints)

---

If you want, I can also produce a **concrete AGENTS.md template**, **requirements.txt contents**, and a **sample golden-output fixture** consistent with the spec, so the developer can start coding immediately.



