Alexander Flood

SNHU CS-499

28th September 2025

**Artifact Two Narrative**

For this enhancement, I extended my Weight Tracker app with two lightweight analytics features, a rolling average over recent entries and a simple linear trend to project when a user might reach their weight goal. Both features operate on the app’s time-series of weight entries (stored as ISO timestamps and values in pounds). The implementation is a windowed rolling average to smooth noise and an ordinary least squares regression to estimate trend in lb/day and a projected “goal date” when the line crosses the goal. All computations are O(n) over the number of records.

I included this enhancement to showcase my ability to design and integrate algorithms and data structures into a mobile application in a way that resembles professional practices as closely as possible. It goes beyond CRUD (create, read, update, and delete) by applying time-series based processing and edge-case handling. The result provides the user value by helping quantify their progress and estimating a goal date. It also demonstrates clear communication through concise comments to improve maintainability.

**Improvements**

I added two analytics, a rolling average to smooth entries and a simple linear regression, to estimate the weight trend over time. The code returns null/NaN in some clearly defined cases, such as fewer than two points, identical dates, or a non-decreasing trend that can’t reach a lower goal. From a software engineering standpoint, all analytics live in a single Analytics utility compose of O(n) functions that have been integrated into the GridActivity.

The UI now summarizes a 7-entry average, slope, and goal projection when appropriate, and the user experience remains responsive by reusing the data that is already fetched instead of making extra network calls. Finally, I documented preconditions, units, return value, and edge cases with concise inline comments and chose method names that fit the style of my project.

**Design choices**

I used an entry-count window (the last 7 entries) rather than a time-based window to keep the implementation simple and predictably O(n). The choice makes it easy to explain to users (“the last 7 logs”) and avoids edge cases when people log irregularly. Using running sums keeps the math straightforward and stable for app-scale datasets. The analytics package can be extended as needed if many users request deeper analytics.

Another important choice was handling the case of a user having fewer than two usable points or their weight loss trend being non-decreasing (weight gain), in which case the app omits the projection and shows a clear message. Using running sums for simple linear regression keeps math stable and fast for the scale of data this app will be handling, and all analytics logic lives in a small Analytics utility class. This separation keeps the UI clean, makes testing straightforward, and leaves room to extend with more in-depth analytics later.

**Challenges**

Implementing these features posed several challenges. First, I had to define safe behavior for cases where data was sparse as well as non-downward trends (cases of weight gain) so the app wouldn’t produce misleading projections. Second, converting the ISO timestamps to epoch days simplified the linear regression math, but only after I fixed a time-zone normalization bug that initially broke projections, which are now based on calendar days.

Finally, integrating the average and projection into the existing UI required rethinking the layout. I overcame this challenge by adding a compact summary above the list that shows the 7-entry average, the slope (lb/day), and the projected goal date (when available). These choices kept the UI responsive and made the feature stand out without clutter.

**Outcomes**

The work on this artifact demonstrates work towards outcome 3 by adding efficient, time-series computations (a rolling average and simple linear regression) implemented in O(n). The code handles edge cases like having too little data or non-decreasing trends by returning null/NaN, ensuring correctness.

The work also supports outcome 4 through implementing the enhancements through clean and module code. All analytics live in a testable analytics utility and are integrated into the GridActivity without coupling or extra network calls. The UI presents results in a compact and easy to read summary and is gracefully hidden when not meaningful.

Finally, it contributes to outcomes 2 by using clear names, concise inline comments, and user facing messages that explain the results and limitations in plain language. This not only increases the usability of the application itself, it also improves the maintainability of the codebase as well.

**Future Work**

As far as future improvements I could make regarding data structures and algorithms would be adding visualizations. I currently display trend and projection information as text above the weight entries, and this information could be made more digestible for users by using a graph. Finally, as a part of my third enhancement, I’ll turn the current SQLite layer into an offline-first cache for the MySQL-backed API. This will allow users to record weights even if their device has lost connection. When connectivity returns, a simple timestamp-based marge will reconcile the differences and refresh the charts to reflect current data.