**Enhancement Two: Data Structures & Algorithms**

**Description of the Artifact**

This enhancement centers around a feature that calculates and visualizes the **average political bias** of all articles saved by a user. It was implemented as part of a React-based web application called *Tilted*, a political bias tracker that helps users analyze the ideological leanings of news content. This feature did **not exist** in the original version of the project (a travel app called *Travlr* built with the MEAN stack during CS-465 Full Stack Development). It is a completely new addition that reflects both functional innovation and algorithmic reasoning.

When users save articles, each article contains a rating\_num value ranging from 1 (left) to 5 (right). The enhancement involves aggregating these values, computing their average, and then transforming the result into a **textual label** (e.g., “center-left”) and a **visual cue** using a rotated red arrow placed over a protractor image. This simple yet effective user feedback mechanism turns numeric data into a meaningful summary.

**Justification for Inclusion**

This enhancement was selected for my portfolio because it demonstrates how data structures and algorithmic thinking can solve practical design problems. Although not overly complex in a computational sense, the feature showcases an ability to:

* Traverse collections (.map, .filter, .reduce) to extract and manipulate relevant data,
* Convert numerical averages into qualitative categories using mapping logic,
* Apply algorithmic thinking to **drive visual outcomes** (e.g., rotating an arrow on a protractor),
* Make data-driven UI decisions in a user-friendly way.

These techniques are widely used in industry for dashboard features, analytics, and dynamic visualizations. The averageRatingNum is calculated from valid entries, rounded to the nearest integer, and mapped to a label using an object lookup. A separate function or switch-case converts the same average into a rotation angle. This algorithm-to-UI bridge illustrates how abstract computation can directly enhance user interaction.

The inclusion of this enhancement shows I can **design modular, reusable functions**, apply **constant-time mappings of O(n))**, and use **React component props and state** to encapsulate logic — all key software engineering and DSA skills.

**Meeting Course Outcomes**

This enhancement directly supports the following course outcome of designing “solutions that solve a given problem using algorithmic principles”. The enhancement required identifying a real problem (how to summarize bias trends), designing a suitable algorithmic solution (aggregation + transformation), and implementing it with **readable, maintainable logic**. The trade-off of computing the average on the client side versus storing a precomputed value was considered, and I chose the dynamic approach for simplicity and flexibility at this stage. I also evaluated alternative implementations — for example, replacing the switch statement used to compute the rotation angle with a constant-time object lookup — and weighed the readability and maintainability of each.

**Reflection on the Process**

During the development of this feature, I gained a deeper appreciation for **algorithmic thinking in UI development**. Although the logic was straightforward, translating it into a reliable, user-friendly component required careful design. I had to account for edge cases (e.g., no saved articles), ensure data integrity (filtering out non-numeric values), and maintain **separation of concerns** between computation and presentation.

One of the most satisfying lessons was learning how a small algorithm — like a reduce function to compute an average — could provide meaningful insight to users. Additionally, visualizing this value via a rotating arrow (using transform: rotate(...)) introduced me to the challenge of aligning algorithmic output with CSS-based visual transformations.

The biggest technical insight came from exploring trade-offs in structure. I learned how to weigh clarity vs. efficiency, such as using .map().filter().reduce() in one pipeline versus pre-filtering or storing values in memory for reuse. I also recognized how simple data structures like **objects as lookup tables** can drastically improve scalability and legibility, especially when paired with predictable keys like bias scores.

In future iterations, this feature could be expanded with a bar chart breakdown of political leanings or bias over time — improvements that could reuse the same foundation of data aggregation and transformation logic.