Roadmap: Multiplier Dashboard & Comparative Analytics Engine

1. Strategic Concept: The "Gap Analysis"

The Multiplier Dashboard will evolve from a static status display into a **Comparative Opportunity Tool**.

- **Goal:** To allow a competitor to identify "Holes in the Swing" by analyzing the "Negative Space"—the multipliers active in the contest that were missed.
- **Mechanism:** A **Three-Way Comparison Loop** (The "Triangulation").
  - *Why Three-Way?* It validates activity. If Station B *and* Station C worked a multiplier that Station A missed, it proves the multiplier was active and workable by the peer group.
- **The "Size of the Prize":** The dashboard must contextualize the *Missed Count* against the *Common Count* to help the user prioritize.
  - *High Common / Low Missed:* High Efficiency (Maintain).
  - *Low Common / High Missed:* Low Efficiency (Urgent Fix Needed).

2. Core Metrics & Logic

The Data Access Layer (DAL) will implement a `ComparativeEngine` (Python) that processes `Sets` of multiplier identities to generate three integers per Band/Mode.

2.1 The "Universe" (Total Activity)

$$\text{Universe} = \text{Log}_A \cup \text{Log}_B \cup \text{Log}_C$$

- **Definition:** Every multiplier worked by *anyone* in the comparison group.
- **Meaning:** The theoretical maximum score available to this specific group of stations.

2.2 The Three Critical Integers

For every Row (Band/Mode) and Column (Mult Type), we calculate:

1. **Common (The Scale):**
   - **Formula:** $\text{Log}_A \cap \text{Log}_B \cap \text{Log}_C$
   - **Value:** Establishes the baseline activity. It acts as the denominator for efficiency.
2. **Differential (The Edge):**
   - **Formula:** $\text{Count}(\text{Log}_A) - \text{Count}(\text{Common})$

- **Value:** Represents the station's "Value Add" relative to the group baseline. This includes strictly unique items and items shared with only one other station.

3. **Missed (The Opportunity):**
   - **Formula:** $\text{Count}(\text{Universe}) - \text{Count}(\text{Log}_A)$
   - **Value: The Primary Metric.** This is the definitive count of active multipliers that the station failed to log.

---

3. Dashboard UX Composition

The design utilizes a **Three-Pane Layout** to balance context with diagnostic depth.

Pane 1: The Context Scoreboards (Top Left)

- **Content:** Condensed "Standard" contest grids for all 2-3 stations side-by-side.
- **Data:** Total QSOs, Zones, Countries, Total Score.
- **Purpose:** Anchoring. It shows *who* won; the next panes show the composition of the score.

Pane 2: The Opportunity Matrix (Top Right / Center)

- **Content:** A high-level Comparative Matrix grouped by Band (CQWW) or Mode (ARRL 10).
- **Visual Priority:** The **Missed Count** (Red/Negative) is the loudest signal.
- **Layout:** | Band | Type | Common (Scale) | Stn A (Diff / Missed) | Stn B (Diff / Missed) | | :--- | :--- | :---: | :---: | :---: | | **20M** | Cty | **110** | +15 / **-2** | +8 / **-12** |
- **Insight:** "The Common count is 110. Stn A missed only 2. Stn B missed 12."
- **Interaction:** Clicking the Red `Missed` number triggers the Drill-Down popup.

Pane 3: Geographic Opportunity Map (Bottom)

- **Content:** A breakdown of the **Total Missed Count** (from Pane 2) by Continent/Region.
- **Logic:** Aggregates the `Missed List` by the entity's continent metadata.
- **Purpose:** Diagnoses *where* the station is failing (e.g., -10 missed in EU).
- **Rows:** NA, SA, EU, AF, OC, AS.
  - *Note:* Japan (JA) is included in AS for standard multiplier dashboards but the architecture supports breaking it out if needed.

Drill-Down Popup (The "Run/S&P" Detail)

- **Trigger:** Click on any "Missed" count in Pane 2.
- **Content:** A list of the specific missed multipliers.
- **Columns:** `Multiplier` | `Band` | `Worked By` | `Method (Run/S&P)`
- **Data Source:** The `Method` is derived from the logs of the stations that *did* work it.
- **Value:** Allows the user to see if the competitor hunted the mult (S&P) or attracted it (Run).

---

4. Technical Architecture (Python DAL)

We will migrate logic *out* of the report generation layer and *into* a reusable Service Layer.

4.1 The Comparative Engine

A generic engine utilizing Python `sets` for high-performance comparison of *any* identifiable object (Multipliers or QSOs).

CODE_BLOCK python from dataclasses import dataclass from typing import List, Set, Dict, Any, Optional

@dataclass class MultiplierInfo: code: str continent: str # 'Run', 'S&P', or 'Both' derived from the competitor logs acquisition_method: Optional[str] = None

@dataclass class StationMetrics: log_id: str worked_count: int # Differential: My Count - Common Count (The "Plus") unique_differential: int # Missed: Universe - My Count (The "Minus") missed_count: int # The actual set of missed objects for Drill-Downs missed_items: Set[MultiplierInfo]

@dataclass class ComparisonResult: universe: Set[Any] common: Set[Any] results: Dict[str, StationMetrics]

class ComparativeEngine: def compare_logs(self, logs: Dict[str, Set[Any]]) -> ComparisonResult: # 1. Universe (Union) universe = set().union(*logs.values())

```
    # 2. Common (Intersection)
    common = set(next(iter(logs.values())))
    for s in logs.values():
        common.intersection_update(s)

    # 3. Station Metrics
    results = {}
    for log_id, log_set in logs.items():
        missed = universe - log_set
        results[log_id] = StationMetrics(
```

```
            log_id=log_id,
            worked_count=len(log_set),
            unique_differential=len(log_set) - len(common),
            missed_count=len(missed),
            missed_items=missed
        )
    return ComparisonResult(universe, common, results)
```

CODE_BLOCK

4.2 Contest Adaptations

The View Logic must request data groupings based on `ContestType`.

- **CQ WW:** Group by **Band** (160, 80, 40...). Rows for Zones and Countries.
- **ARRL 10 Meter:** Group by **Mode** (CW, PH). Rows for State, Province, Mexico, DX.
- **CQ WPX:** Group by **Band**. Single Row for "Prefix".
  - *Special Handling:* Due to high volume, Drill-Downs for WPX may be disabled or limited to "Top 10 Missed," but the Matrix Integers (Common vs Missed) remain valid indicators of efficiency.

---

5. Implementation Roadmap

Phase 1: DAL Core (Python)

- **Task:** Implement `ComparativeEngine` class and `ComparisonResult` dataclasses.
- **Validation:** Unit tests using the specific counts from the provided CQ WW text reports to ensure the "Missed" math is exact.

Phase 2: Metadata & Grouping

- **Task:** Implement the "Grouper" logic. The DAL must accept raw CSV rows and group them into Sets based on the Contest Rules (e.g., `Mults_By_Band` vs `Mults_By_Mode`).
- **Task:** Ensure Multiplier objects contain `Continent` metadata (for Pane 3) and `Run/S&P` metadata (for Drill-Downs).

Phase 3: Dashboard View (The Matrix)

- **Task:** Build the UI for Pane 2 using a Grid/Table component.
- **Task:** Implement the "Common / Diff / Missed" cell renderer.
- **Constraint:** Must handle dynamic columns (comparing 2 logs vs 3 logs).

Phase 4: Geographic Map & Drill-Down

- **Task:** Build Pane 3 (Geo Table).
- **Task:** Implement the "Missed List" popup triggered by clicking the Red integers in Pane 2, ensuring the `Method` column is populated.