Building an Assessment-Ready Conduct Data Model

Transforming Maxient Exports into Reliable Dashboards for Student Conduct Analysis

Methodology and Data Model Documentation

Joshua L. Moermond

Assistant Director for Residential Conduct and Assessment

Student Conduct and Community Standards

University of Cincinnati

Last updated: January 20, 2026

Contents

1 Executive Summary

The Problem: Student conduct data from Maxient is designed for case management, not analysis. When practitioners try to answer questions like "Are alcohol incidents increasing?" or "Which halls need more support?", they find exports that are inconsistent across years, full of duplicate spellings, and structured in ways that make accurate counting nearly impossible.

The Solution: This project transforms raw Maxient exports into clean, organized tables that Power BI can use reliably. The result is dashboards where filters work as expected, numbers stay stable, and stakeholders can trust what they're seeing.

These dashboards are designed to support **aggregate analysis and strategic conversation**, not case-level decision-making or individual evaluation.

What You Can Now Do:

- Compare trends across academic years without manual data cleanup
- Filter by hall, term, case type, or staff assignment and get consistent counts
- Calculate violation rates per 100 residents for fair comparisons across buildings
- Track how long cases take to resolve, adjusted for breaks and holidays
- Share dashboards more broadly because student identifiers have been removed
- Use aggregate trends and rates to inform policy discussions, resource planning, and prevention strategies—while grounding interpretation in professional judgment and institutional context

Who Should Read What:

If you are...	Focus on...	You can skip...
Senior Leadership	Executive Summary and Common Questions this Model can Answer	Everything else
Unit Directors, Deans	Executive Summary, Section 4 (Common Questions), callout boxes throughout	Code blocks, technical details
Conduct Professionals	Sections 1-4, table descriptions in Section 5	Code blocks

If you are…	Focus on…	You can skip…
Analysts and IT Professionals	Everything, especially code blocks and Section 6 (Computing Environment)	Nothing

This document is intended to support informed discussion and assessment; it is not a compliance report or performance evaluation tool.

2 Accessibility Statement

This document has been designed with accessibility in mind and strives to meet Web Content Accessibility Guidelines (WCAG) 2.1 Level AA standards. However, due to current limitations in LaTeX-based PDF generation technology, some accessibility features require post-processing:

Known Limitations:

- **PDF Tagging**: LaTeX does not natively generate tagged PDFs with full semantic structure. This document will undergo automatic tagging in Adobe Acrobat Pro to ensure proper document structure for assistive technologies.
- **Table Headers**: While all tables in this document use proper header rows, LaTeX-generated PDFs may not automatically mark them as header cells.
- **Code Blocks**: R code chunks are rendered as syntax-highlighted text but may be interpreted by PDF readers as images or figures. During accessibility review, these are marked as decorative. Descriptive text accompanies each code section to ensure the methodology is accessible regardless of code readability.

Commitment to Accessibility:

The author is committed to making this document fully accessible. If you encounter any accessibility barriers or need this document in an alternative format, please contact:

- Joshua L. Moermond
- Email: moermondsahe@gmail.com

An HTML version of this document is available in the project repository on Github may provide better compatibility with some assistive technologies.

3 Introduction

Student conduct systems like Maxient are built to help staff manage cases—documenting incidents, tracking outcomes, and ensuring follow-up. They do this well. But when it comes time to step back and ask bigger questions—*Are we seeing more alcohol incidents this year? Which halls need more support? How long do cases typically take to resolve?*—the data becomes difficult to use.

This isn't because anyone is doing something wrong. It's because case management systems and analytics systems have different goals. Case management needs flexibility; analytics needs consistency. As a result, practitioners often hit familiar walls:

- Exports look different from year to year, making comparisons difficult.
- The same location or violation might be spelled three different ways.
- A single export row might contain the incident, the charges, the outcome, and the sanctions all at once—making it hard to count any one thing cleanly.
- Privacy concerns limit what can be shared or who can access the data.

This project solves those problems by transforming raw Maxient exports into a **dashboard-ready data model** designed for Power BI. The process:

1. Combines multiple years of data into one consistent dataset,
2. Standardizes how key fields are recorded (dates, locations, roles, outcomes),
3. Adds academic calendar context (academic year, semester, week),
4. Calculates useful process measures (like how long cases take to resolve),
5. Scrambles identifiable information using a secure code so it can't be traced back to real students, and
6. Organizes everything into a structure that makes dashboards reliable.

3.1 What Is a Star Schema?

Think of a star schema as an organized filing system where related information connects like a hub-and-spoke wheel.

At the center (the hub) are tables storing **what happened**—incidents, violations, sanctions, case timelines. These are called "fact tables."

Radiating outward (the spokes) are tables storing **context about what happened**—when it occurred, where, who was involved, what type of case. These are called "dimension tables."

Why does this matter? When these two types of information are mixed together in one giant spreadsheet, dashboards can give inconsistent answers. Filtering by one thing accidentally changes the count of something else. Numbers shift depending on how you slice the data. People stop trusting the reports.

A star schema prevents this by keeping "what happened" and "context" in separate but connected tables. The result: filters work as expected, counts stay stable, and stakeholders can trust what they're seeing.

Why This Matters

For Decision-Makers: Dashboards become trustworthy. When you filter by residence hall or academic year, the numbers won't mysteriously change based on what else is on the page.

For Analysts: This is the structure Microsoft recommends for PowerBI. It improves query performance and ensures DAX measures calculate correctly across filter contexts. A full entity-relationship diagram of the model is available in Appendix A.

The ultimate goal is to support better decisions in student conduct work:

- **Staffing**: Where is the workload concentrated? Do we need more support in certain areas or at certain times?
- **Policy**: Are our policies producing the outcomes we expect? What happens after we make changes?
- **Equity**: Are patterns consistent across student populations and environments, or are there disparities worth investigating?
- **Trends**: How is this year different from last year? Are changes meaningful or just noise?

3.2 How to Read This Document

3.2.1 About the Code Blocks

Throughout this document, you'll see gray boxes containing R code. These show *how* each table is created behind the scenes. They're included for transparency and so the process can be repeated exactly the same way each time.

If you're not familiar with R or data analytics, **you can skip the code entirely** and still understand what the model does and why it matters.

Why Include Code?

For Decision-Makers: You don't need to read it—but knowing it exists means this process doesn't depend on one person's memory. If Joshua wins the lottery and moves to Tahiti, someone else can pick this up.

For Analysts: The code is the documentation. You can trace exactly what happens to each field, verify the logic, and adapt it for new requirements.

3.2.2 About the `df_schema()` Outputs

Starting in Section 5, you'll see small tables that show the structure of each dataset. These outputs show:

- The **column names** (what information is captured)
- The **data type** of each column (date, text, number, etc.)

These outputs do *not* show actual data—no student names, no case details. They answer the question: *"What does this table look like?"*

This helps readers quickly understand what information is available in each table and how tables relate to each other.

Why Structure Matters

For Decision-Makers: Knowing what fields exist helps you ask better questions. If you see "INCIDENT_LOCATION" in the table, you know you can filter by building.

For Analysts: Understanding the schema prevents errors. For example, knowing that FactViolation counts individual charges (not cases) helps you avoid accidentally reporting the wrong number.

3.3 Scope and Non-Goals

This project is designed to support **program-level assessment, strategic decision-making, and institutional learning** in student conduct and student affairs contexts. It focuses on transforming operational case-management data into a structure that supports reliable aggregation, trend analysis, and comparison over time.

This model is **not intended** to:

- Serve as a real-time operational monitoring or case-management tool
- Replace case-level review, professional judgment, or due process
- Establish causal relationships between policies, interventions, and outcomes
- Function as a performance evaluation tool for individual staff members
- Generalize directly to other institutions without local adaptation

While the transformation pipeline produces stable and reproducible metrics, interpretation of those metrics must always be grounded in institutional context, policy intent, and professional practice.

4 Definitions & Reporting Concepts

This section defines key terms used throughout the document. You don't need to memorize these—they're here as a reference when you encounter unfamiliar terminology.

4.1 The One Concept That Prevents Most Reporting Errors

Grain answers the question: *What does one row in this table represent?*

This single concept prevents most reporting disagreements. When two reports show different numbers, the first question should always be: *What are we counting?*

Think of it like a receipt. A grocery store receipt has one line per item purchased—that's the grain. If you count lines on your receipt, you get the number of items, not the number of shopping trips or the number of stores.

In conduct data:

Table	What one row represents	If you count rows, you get...
FactIncident	One incident record	Number of incidents
FactViolation	One alleged charge	Number of charges filed
FactSanction	One sanction applied	Number of sanctions given
DimStudent	One unique student	Number of students in the system

Why this matters: A single incident might involve multiple violations and result in multiple sanctions. A student involved in three incidents appears in three rows of FactIncident but only one row in DimStudent.

If someone asks "How many students were involved in incidents last year?" and you count rows in FactIncident, you'll overcount students who had multiple incidents. You need to count distinct students instead—or use DimStudent.

The Fastest Way to Resolve Number Disagreements

For Decision-Makers: When two reports show different numbers, ask: "What are we counting—cases, incidents, violations, sanctions, or students?" The answer usually explains the discrepancy.

> **For Analysts:** Always document the grain of any metric you create. "Incident count" and "student count" can differ by 30% or more depending on repeat involvement rates.

4.2 Fact Tables vs. Dimension Tables

4.2.1 Fact Tables: What Happened

A fact table stores **events and measures**—the things you want to count or analyze. Each row represents something that occurred.

In this model:

Fact Table	What each row represents	Example questions it answers
FactIncident	One incident	"How many incidents occurred in Fall 2024?"
FactViolation	One alleged charge	"What's the not-responsible rate for alcohol violations?"
FactSanction	One sanction applied	"How often do we assign educational programs vs. probation?"
FactTimeline	Timing for one case	"How long do cases take from report to resolution?"
FactRecidivism	Repeat involvement summary by year	"What percentage of students have multiple cases within an academic year?"
FactCohortRecidivism	One student (by first responsible finding)	"Of students who first got in trouble in AY2223, how many ever got in trouble again?"
FactHousingCensus	Violations + population for one hall/term	"Which hall has the highest violation rate per 100 residents?"
FactDeadline	One case deadline	"How many cases are currently overdue?"

4.2.2 Dimension Tables: Context About What Happened

A dimension table stores **descriptive information** that helps you filter and interpret events.

In this model:

Dimension Table	What it describes	Example use
DimDate	Calendar and academic time	Filter by semester, compare academic years
DimAcademicYear	Academic year boundaries	Ensure AY2324 means Aug 2023–Jul 2024 everywhere
DimAcademicTerm	Fall/Spring/Summer terms	Compare Fall-to-Fall trends
DimHousing	Residence hall attributes	Filter by building type or ownership
DimHousingYear	Hall attributes that change yearly	Track capacity changes over time
DimStudent	Student demographics/academics	Disaggregate by classification, college, ethnicity
DimCase	Case-level attributes	Filter by case type, status, assigned officer
DimSanction	Sanction categories and severity	Group sanctions by type (educational, restrictive, etc.)
DimCollege	Academic colleges	Filter or group by college
DimHearingOfficer	Staff assignments	Analyze workload by officer or department
DimViolation	Charges metadata (categories, severity, etc.)	Group charges by type and severity

Why Separate Facts from Context?

For Decision-Makers: This separation means that when you filter by residence hall or semester, you change *which* incidents are included—but you don't change *what* "incident" means. The definition stays stable.

For Analysts: In Power BI terms, this creates clean one-to-many relationships. Dimensions filter facts, not the reverse. This prevents circular dependencies and ambiguous filter propagation.

4.3 Counts vs. Rates

A **count** is the total number of events: *"40 violations in Calhoun Hall."*

A **rate** compares a count to a population: *"5 violations per 100 residents."*

Counts describe volume, not severity, intent, or community climate, and should never be interpreted as measures of "how problematic" a group or location is.

When to use each:

Question Type	Use	Example
"How much work is this?"	Count	"We had 200 cases this semester—do we need more staff?"
"Where should we focus prevention?"	Rate	"Building A has 12 violations per 100 residents vs. Building B's 4 per 100"
"Is this building 'worse' than that one?"	Rate	Counts alone can mislead when populations differ

Example:

Hall	Violations	Residents	Rate per 100
Calhoun	50	800	6.25
Dabney	25	200	12.50

Calhoun has twice as many violations, but Dabney's *rate* is twice as high. Which number matters depends on your question:

- For staffing Calhoun (workload), the count matters.
- For targeting prevention efforts (where is behavior concentrated?), the rate matters.

Fair Comparisons Require Rates

For Decision-Makers: When comparing halls, terms, or populations of different sizes, always ask for the rate, not just the count. A hall with higher counts may simply have more residents.

For Analysts: FactHousingCensus pre-calculates violation rates per 100 residents by hall and term, joining violation counts with census data.

4.4 Time Intelligence: Academic vs. Calendar Time

Student Affairs work follows the academic calendar, not the calendar year. "Last year" usually means "last academic year" (August to July), not "last calendar year" (January to December).

This model includes a dedicated date table (DimDate) that ensures:

- **"AY2324" means the same thing everywhere**: August 1, 2023 through July 31, 2024
- **Semester comparisons are consistent**: Fall is always August–December; Spring is always January–April
- **Year-over-year trends align properly**: Comparing Fall 2023 to Fall 2022, not December 2023 to December 2022

The related tables DimAcademicYear and DimAcademicTerm provide additional structure for filtering and grouping by academic periods.

Why Academic Time Matters

For Decision-Makers: Reports that use calendar years can obscure patterns. A spike in "January 2024" might be the end of Fall 2023's cases being resolved, not a

Spring 2024 trend.

For Analysts: DimDate includes an AcademicTermKey that links to DimAcademicTerm, enabling correct semester-level aggregation. The AY_Order field ensures academic years sort correctly (AY2223 before AY2324).

5 Methodology

This section describes *how* the transformation works at a conceptual level. If you want to understand the approach without reading code, this section is for you. The actual implementation details appear in Section 5.

5.1 Overview

5.1.1 Five Steps from Messy to Usable Conduct Data

1. **Combine**
   Multiple Maxient exports are merged into a single, unified dataset so that trends can be analyzed consistently across academic years.
2. **Standardize**
   Inconsistent values (e.g., *CALHOUN*, *Calhoun Hall*, *calhoun*) are normalized to a single standard representation, ensuring that identical concepts are always treated the same way.
3. **Add Academic Time**
   Raw date fields are transformed into academic-year, semester, and week-level variables so that reporting aligns with how Student Affairs work is organized and evaluated.
4. **Calculate Process Metrics**
   Key timeline metrics—such as days from report to case creation and resolution—are computed to identify bottlenecks, monitor efficiency, and support assessment.
5. **Protect Privacy and Organize for Analysis**
   Direct identifiers are anonymized, and the dataset is reshaped into a star schema structure so dashboards can be shared more broadly while remaining reliable and privacy-conscious.

Each step addresses a specific barrier that makes raw Maxient data difficult to use for assessment.

5.2 Step 1: Combine Multiple Years

The Problem: Maxient exports are limited in size, so multi-year data requires multiple export files. These files may have slightly different column structures depending on when they were created or what system updates occurred.

The Solution: The pipeline reads all export files and merges them into a single dataset, handling any column differences automatically.

Why This Matters: Longitudinal analysis—comparing this year to last year, or tracking five-year trends—requires data to be in one place. Without this step, every trend analysis would require manual file merging.

> *Why Combine Data?*
>
> **For Decision-Makers:** You can now answer questions like "How has incident volume changed over five years?" without anyone manually copying and pasting spreadsheets together.
>
> **For Analysts:** The pipeline uses `bind_rows()` which handles column mismatches gracefully—columns present in some files but not others become NA values rather than causing errors.

5.3 Step 2: Standardize Key Fields

The Problem: The same information gets recorded differently over time and across users:

Field	Variations Found	Standardized To
Location	"CALHOUN," "Calhoun Hall," "Calhoun"	"Calhoun Hall"
Location	"Stratford Hts Bld 2," "Stratford Heights Bldg 2"	"Stratford Heights"
Ethnicity	"BLACK," "Black NotHisp," "African-American"	"Black"
Role	"Alleged/Respondent," "Respondent"	"Respondent"

Field	Variations Found	Standardized To
Sport	"M-Football," "Football," "Men's Football"	"Men's Football"

The Solution: The pipeline applies consistent mappings that convert all variations to a single standard value. It also:

- Converts date fields to proper date format (not text)
- Replaces blank values with "Not Reported" so they show up in filters instead of disappearing
- Recodes obsolete categories (e.g., "Pre-Junior" → "Sophomore")

Why This Matters: If "Calhoun Hall" and "CALHOUN" are treated as different places, your counts are wrong. Standardization ensures that when you filter by a location, you capture all relevant records.

Standardization improves consistency for analysis, but it does not eliminate professional judgment or contextual nuance in how cases are documented and interpreted.

Why Standardization Protects Your Analysis

For Decision-Makers: You can trust that "Calhoun Hall" in the dashboard means *all* Calhoun Hall incidents, not just the ones that happened to be spelled a certain way.

For Analysts: Standardization happens once, upstream, in the pipeline. This means every downstream table and dashboard inherits clean values without needing its own cleanup logic. The mappings are documented in `config.R` and can be updated as new variations appear.

5.4 Step 3: Add Academic Time Structure

The Problem: Maxient stores dates as calendar dates, but Student Affairs work is organized around the academic calendar. Questions like "How does this semester compare to last semester?" or "What happened during Fall 2023?" require academic time fields that don't exist in the raw data.

The Solution: The pipeline derives academic time fields from incident dates:

Derived Field	What It Captures	Example
ACADEMIC_YEAR	Aug 1 – Jul 31 cycle	"AY2324" (Aug 2023 – Jul 2024)
Semester	Term within the year	"FA" (Fall), "SP" (Spring), "SU" (Summer)
WEEK_NUM	ISO week number	Week 1–52 for week-over-week patterns
incident_month	Month, ordered Aug–Jul	Allows academic-year month trends
incident_wday	Day of week	For day-of-week patterns

Why This Matters: A report showing "January 2024 had high case volume" might be misleading—that could be Fall 2023 cases reaching resolution, not a Spring 2024 trend. Academic time fields let you interpret patterns in context.

Why Academic Time Changes Everything

For Decision-Makers: You can now ask "How does Fall 2024 compare to Fall 2023?" and get a meaningful answer. Calendar-year reports often obscure the patterns that matter for academic operations.

For Analysts: The DimDate table provides a complete date spine with pre-calculated academic year, semester, and term keys. Fact tables join to DimDate on incident date, enabling time intelligence in Power BI without complex DAX calculations.

5.5 Step 4: Calculate Process Metrics

The Problem: Understanding workflow efficiency requires knowing how long cases take to move through the process. But simply subtracting dates can produce misleading results:

- A case reported December 15 and resolved January 15 looks like 31 days—but the office was closed for winter break for 20 of those days.

- A case might show negative days if dates were entered incorrectly.

The Solution: The pipeline calculates three timing metrics:

Metric	What It Measures	Calculation
reported_to_case	Intake/triage speed	Days from report date to case creation date
case_to_resolution	Adjudication speed	Days from case creation to hearing/resolution date
reported_to_resolution	Total process time	Days from report to resolution

Each calculation:

- **Subtracts institutional pause periods** (breaks, holidays) so a case that spans winter break isn't penalized for closed days
- **Floors negative values to zero** so data entry errors don't produce impossible results

Why This Matters: Timing metrics help identify where delays occur. Is the bottleneck at intake (getting cases created) or adjudication (getting cases resolved)? Are certain case types systematically slower?

How to Interpret Timing Metrics

For Decision-Makers: Longer timelines aren't automatically "bad." They might reflect case complexity, due process requirements, student availability, or staffing patterns. Use timing data as a *signal to investigate*, not a performance score.

For Analysts: Pause periods are defined in `config.R` as `OMIT_RANGES`. The `compute_overlaps()` function calculates how many days of each case's timeline fall within these ranges and subtracts them. Warning Letter cases are excluded from adjustment since they resolve immediately.

5.6 Step 5: Protect Privacy and Organize for Reporting

The Problem: Raw Maxient exports contain student IDs, case numbers, and other identifiable information. Sharing dashboards broadly—with housing partners, leadership, or

for accreditation—creates privacy risk. Additionally, the wide format of Maxient exports (one row per case with many columns) doesn't work well for dashboard tools.

The Solution: The pipeline does two things: protects privacy and organizing data for dashboard-ready reporting.

5.6.1 Privacy Protection

Identifiable fields (case numbers, file IDs, student IDs) are scrambled using a secure code. The scrambled values:

- **Look like random text** (e.g., "a3f7c9e2b1d8...")
- **Are consistent** (the same student ID always produces the same scrambled value, so records can still be linked across tables)
- **Cannot be reversed** (you can't work backward from the scrambled value to find the real ID)

This means dashboards can show patterns involving individual students (like repeat involvement) without exposing who those students are.

5.6.2 Star Schema Organization

The wide Maxient export is split into separate fact and dimension tables:

Before (one wide row per case):

```
FILE_ID | Student_Name | DOB | Hall | Charge_1 | Charge_2 | Finding_1 | ...
```

After (separate tables linked by keys):

```
FactIncident: FILE_ID | INCIDENT_DATE | INCIDENT_LOCATION | ...
FactViolation: FILE_ID | VIOLATION | OUTCOME | ...
FactSanction: FILE_ID | Sanction | ...
DimStudent: SID | ETHNICITY | CLASSIFICATION | ...
DimHousing: INCIDENT_LOCATION | Housing_Type | ...
```

This separation means:

- Each table has a clear grain (what one row represents)

- Descriptive information isn't duplicated across thousands of rows
- Power BI can filter efficiently because relationships are explicit

Why This Enables Broader Access

For Decision-Makers: With identifiable information removed, dashboards can be shared with more stakeholders—housing partners, senior leadership, accreditation reviewers—without FERPA concerns. Patterns are visible; individuals are not.

For Analysts: The scrambling uses HMAC-SHA256 with a locally stored key. The same key must be used each time to ensure consistency across refreshes. The key file (`pepper.bin`) should be protected and backed up but never shared or committed to version control.

5.7 Data Sources

5.7.1 Primary Source: Maxient Case Exports

The pipeline ingests Maxient case exports covering multiple academic years. These exports are generated from Maxient's reporting tools and contain:

- Case and incident information
- Student demographic data (as recorded in Maxient)
- Charges and findings
- Sanctions applied
- Assignment and resolution information

Multiple exports are required because:

1. Maxient is a web-based tool and doesn't perform well with large exports
2. Historical data may need to be re-exported after system updates
3. Different date ranges may have been exported at different times

5.7.2 Supplementary Sources: Lookup Tables

Several external lookup tables enrich the Maxient data:

File	Purpose	Source
`academic_plans.csv`	Maps major codes to college names	Registrar
`hearing_officers.csv`	Maps assignment codes to staff names/departments	SCCS
`DimHousing.csv`	Housing attributes (type, ownership, room style)	University Housing
`DimHousingYear.csv`	Year-specific housing attributes	University Housing
`DimSanction.csv`	Sanction categories and severity classifications	SCCS
`housing_census.csv`	Resident counts by building and term	University Housing
`DimCharge.csv`	Charge categories, severity, and flags	SCCS

These lookup tables allow the dashboard to answer questions that Maxient data alone cannot—like "What's the violation rate per 100 residents?" or "How do patterns differ by academic college?"

Why Supplementary Data Matters

For Decision-Makers: Raw conduct data tells you *what happened*. Adding context (like building populations or college affiliations) tells you *what it means*. A hall with 50 violations could be a problem or unremarkable depending on how many students live there.

For Analysts: Lookup tables should be updated annually or when organizational changes occur (new buildings, renamed colleges, staff turnover). The pipeline will still run with outdated lookups, but unmatched values will appear as "Not Reported" in dimensions.

5.8 Output: What the Pipeline Produces

The final output is an Excel workbook (`REDConduct_StarSchema.xlsx`) containing all dimension and fact tables as separate worksheets. This workbook is designed for direct import into Power BI.

In Power BI, the workbook is imported using Get Data → Excel. Relationships between tables are established based on shared key fields (FILE_ID, SID, INCIDENT_DATE, INCIDENT_LOCATION, etc.).

The resulting data model supports dashboards for:

- **Volume and trends**: Incident counts over time, by location, by type
- **Workload analysis**: Cases per officer, timeline distributions
- **Outcome patterns**: Finding rates by violation type, sanction frequencies
- **Equity review**: Disaggregation by student demographics
- **Housing context**: Violation rates per 100 residents by hall
- **Process monitoring**: Overdue cases, timeline bottlenecks

6 Data Quality and Reporting Limitations

As with any assessment effort based on administrative data, the outputs of this model reflect both the strengths and limitations of the underlying source systems and reporting practices.

Key considerations include:

1. **Reporting Bias**
   Conduct data reflects what is reported and documented, not all behavior that occurs. Changes in reporting practices, staff training, or institutional priorities may affect trends independent of student behavior.
2. **Charging Discretion**
   Violations and charges reflect professional judgment at the time of case intake. Differences in charging practices across staff, time periods, or policy regimes may influence counts and rates.
3. **Historical Inconsistency**
   Maxient exports may vary across years due to system updates, form changes, or evolving definitions. The pipeline standardizes these differences where possible, but historical context remains important.
4. **Missing or Evolving Data**
   Some fields may be incomplete, inconsistently populated, or newly introduced. Explicit "Not Reported" values are used to preserve transparency rather than silently excluding records.
5. **Small-Number Instability**
   Disaggregated analyses (e.g., by specific buildings, demographics, or niche case types) may be sensitive to small counts. Apparent changes may reflect random variation rather than meaningful shifts.

These limitations do not invalidate the analysis. Instead, they underscore the importance of using this model as a **decision-support tool**, interpreted alongside professional expertise, qualitative context, and institutional knowledge.

7 Common Questions This Model Can Answer

This section translates the data model into practical questions. If you're wondering "What can I actually *do* with this?"—this section is for you.

The questions are organized by use case. For each category, you'll find example questions, which tables provide the answers, and guidance on interpretation.

7.1 Intended Use and Interpretive Guardrails

This model is designed to support **aggregate analysis, trend identification, and strategic conversation** about student conduct systems and outcomes.

It is **not intended** to be used for:

- Individual case adjudication
- Compliance determinations
- Individual staff performance evaluation or comparison
- Automated decision-making or risk scoring

Metrics produced by this model should be used to **ask better questions**, identify areas for further review, and inform policy, training, and resource allocation—not to substitute for professional judgment or contextual understanding. Case-level and staff-assignment fields are included to support **aggregate workload understanding**, not comparative evaluation of individual performance.

7.2 Quick Reference: Which Table Answers Which Question?

If you're asking about...	Start with...	Grain (what one row represents)
How many incidents occurred	FactIncident	One incident record
Violation types and outcomes	FactViolation	One alleged charge
Sanctions and interventions	FactSanction	One sanction applied
How long cases take	FactTimeline	One case's timing metrics
Repeat student involvement	FactRecidivism	Summary by academic year

If you're asking about...	Start with...	Grain (what one row represents)
Violations adjusted for population	FactHousingCensus	One hall in one term
Overdue or pending cases	FactDeadline	One case deadline
Student characteristics	DimStudent	One unique student
Building attributes	DimHousing / DimHousingYear	One building (or building-year)
Staff workload	DimCase + DimHearingOfficer	One case

7.3 Volume and Trend Questions

These questions help you understand *how much* is happening and *whether patterns are changing*.

Example Questions:

- How many incidents occurred this academic year compared to last year?
- Are incident counts increasing, decreasing, or stable over the past five years?
- Which months have the highest incident volume? Is this consistent year to year?
- Are trends similar across all residence halls, or concentrated in specific buildings?
- How does Fall semester compare to Spring semester?

Where to Look:

Question Focus	Primary Table	Key Fields
Incident counts over time	FactIncident	INCIDENT_DATE, joined to DimDate
Counts by location	FactIncident	INCIDENT_LOCATION
Counts by case type	FactIncident	TYPE
Term-over-term comparison	FactIncident	Joined to DimAcademicTerm

Interpretation Guidance

For Decision-Makers: Volume trends alone don't tell you if things are "better" or "worse." A decrease in incidents could mean fewer problems, fewer reports,

or a policy change. Pair volume trends with context: Did occupancy change? Did reporting processes change? Did we add or remove buildings from scope?

For Analysts: When comparing years, ensure you're comparing complete periods. If it's October 2024, comparing "AY2425 so far" to "all of AY2324" will be misleading. Use DimDate to filter to equivalent periods (e.g., Aug 1–Oct 15 of each year).

7.4 Charging and Outcome Questions

These questions help you understand *what violations are being alleged* and *what findings result*.

Example Questions:

- What are the most common violation types?
- Has the mix of violation types changed over time?
- What is the "not responsible" rate overall? By violation type?
- Are we charging more violations per incident than in prior years?
- Do outcome rates differ by case type (e.g., residential vs. academic)?

Where to Look:

Question Focus	Primary Table	Key Fields
Violation frequency	FactViolation	VIOLATION
Outcomes by violation	FactViolation	VIOLATION, OUTCOME
Violations per incident	FactViolation + FactIncident	Count violations, count incidents, divide
Outcomes by case type	FactViolation	TYPE, OUTCOME

Understanding Violations Per Incident:

This metric helps distinguish behavior patterns from charging patterns:

Scenario	What It Might Mean
Violations per incident increasing	Staff may be charging more thoroughly, or incidents may be more complex

Scenario	What It Might Mean
Violations per incident decreasing	Staff may be charging more conservatively, or incidents may be simpler
High not-responsible rate for specific violation	Evidence standards may be difficult to meet, or the violation may be over-charged

Interpretation Guidance

For Decision-Makers: Outcome differences can reflect many things: behavior, evidence quality, policy clarity, charging patterns, or hearing officer interpretation. A high not-responsible rate for a specific violation isn't automatically a problem—it might mean staff are appropriately bringing close cases forward. Use outcome patterns as *prompts for conversation*, not conclusions.

For Analysts: The OUTCOME field contains the finding (Responsible, Not Responsible, etc.). Some violations may have NULL outcomes if the case is still pending. Filter to resolved cases when calculating outcome rates: `filter(!is.na(OUTCOME))` or equivalent in DAX.

7.5 Sanction and Intervention Questions

These questions help you understand *what we do in response* to conduct incidents.

Example Questions:

- Which sanctions are most frequently applied?
- How has our sanction mix changed over time?
- How often do we use educational interventions vs. status sanctions (probation, suspension)?
- Do sanction patterns differ by violation type or case type?
- After adding a new educational program, how often is it being assigned?

Where to Look:

Question Focus	Primary Table	Key Fields
Sanction frequency	FactSanction	Sanction
Sanctions over time	FactSanction	INCIDENT_DATE joined to DimDate

Question Focus	Primary Table	Key Fields
Sanction categories	FactSanction + DimSanction	Join on Sanction to get CATEGORY_LABEL, SEVERITY_LEVEL
Sanctions by case type	FactSanction	TYPE

Understanding Sanction Categories (from DimSanction):

Category	Examples	Purpose
Educational	Online modules, reflection papers, seminars	Learning and behavior change
Status	Probation, reprimand	Formal record without restriction
Restrictive	Loss of privileges, no-contact orders	Limit access or behavior
Removal	Suspension, housing termination	Separation from community
Referral	CAPS consultation, academic advisor	Connection to support services

Interpretation Guidance

For Decision-Makers: Sanctions represent your intervention philosophy in action. If you believe in educational approaches but 80% of sanctions are probation, there may be a gap between philosophy and practice—or probation may be serving an educational purpose in your context. Use sanction data to prompt training conversations and resource planning.

For Analysts: FactSanction has one row per sanction applied. A single case can have multiple sanctions (e.g., probation + educational program). To count cases with any educational sanction, you'll need to aggregate: count distinct FILE_ID where Sanction is in the educational category.

7.6 Process and Timeline Questions

These questions help you understand *how efficiently* cases move through the system.

Example Questions:

- How long do cases typically take from report to resolution?
- Where do delays occur—at intake (report to case creation) or adjudication (case creation to resolution)?
- Are timelines getting longer or shorter over time?
- Do certain case types take longer than others?
- Are there seasonal patterns (e.g., cases taking longer during busy periods)?

Where to Look:

Question Focus	Primary Table	Key Fields
Overall timeline	FactTimeline	reported_to_resolution
Intake speed	FactTimeline	reported_to_case
Adjudication speed	FactTimeline	case_to_resolution
Timelines by case type	FactTimeline + FactIncident	Join on FILE_ID to get TYPE
Timelines over time	FactTimeline	INCIDENT_DATE joined to DimDate

Understanding the Three Timeline Metrics:

Metric	What It Measures	Influenced By
reported_to_case	How quickly we create cases from reports	Intake staffing, triage process, reporting clarity
case_to_resolution	How quickly we resolve created cases	Hearing officer availability, case complexity, student scheduling
reported_to_resolution	Total time in system	All of the above

Interpretation Guidance

For Decision-Makers: Longer timelines aren't automatically bad. A complex Title IX case *should* take longer than a simple noise violation. What matters is whether timelines are *appropriate* for the case type and *consistent* across similar cases. Look for outliers and investigate: Why did this case take 90 days when similar cases take 14?

> **For Analysts:** Timeline metrics have already been adjusted for institutional pause periods (breaks, holidays) as defined in `config.R`. Negative values have been floored to zero. Warning Letter cases skip timeline adjustment since they resolve immediately. When reporting averages, consider using median instead of mean—timeline distributions are typically right-skewed with some very long cases.

7.7 Housing Context and Rate Questions

These questions help you understand conduct patterns *in relation to where students live* and *how many students live there*.

Example Questions:

- Which residence halls have the most violations?
- Which halls have the highest violation *rate* per 100 residents?
- Are violation rates higher in certain building types (traditional vs. suite-style)?
- How have rates changed in a specific building over time?
- If we implemented a prevention program in a hall, did rates change afterward?

Where to Look:

Question Focus	Primary Table	Key Fields
Violations per hall	FactViolation	INCIDENT_LOCATION
Violation rates	FactHousingCensus	VIOLATIONS, CENSUS, Violation_Rate
Building characteristics	DimHousing	Housing_Type, Ownership, Room_Style
Year-specific context	DimHousingYear	Attributes that change annually
Rate trends over time	FactHousingCensus	Academic_Year, Semester

Understanding FactHousingCensus:

This table pre-calculates violation rates so you don't have to:

Field	What It Contains
INCIDENT_LOCATION	Building name
Academic_Year	AY2324, etc.
Semester	FA or SP (Summer excluded due to low census)
CENSUS	Number of residents that term
VIOLATIONS	Count of violations that term
Violation_Rate	(VIOLATIONS ÷ CENSUS) × 100

Interpretation Guidance

For Decision-Makers: Rates enable fair comparisons. A large hall will almost always have more violations than a small hall—that's just math. Rates tell you whether a building has *disproportionate* activity given its size. However, rates can be unstable in small buildings: a hall with 50 residents and 5 violations (10%) could drop to 2% next year just by chance.

For Analysts: FactHousingCensus only includes Fall and Spring terms because summer census is often too small for meaningful rates. If a building has zero census (closed, not yet open), Violation_Rate will be NULL, not zero—this prevents divide-by-zero errors and avoids implying "no violations" when the building wasn't occupied.

7.8 Equity and Disaggregation Questions

These questions help you understand whether conduct patterns *differ across student populations*.

Example Questions:

- Are certain student groups over-represented in conduct incidents relative to their enrollment?
- Do outcome rates (responsible/not responsible) differ by student demographics?
- Are sanction patterns consistent across student groups, or are some groups receiving more severe sanctions?
- Do timeline lengths differ by student demographics?

Where to Look:

Question Focus	Primary Table	Key Fields
Student demographics	DimStudent	ETHNICITY, GENDER, CLASSIFICATION
Academic context	DimStudent	College, Major
Involvement by group	FactIncident + DimStudent	Join on SID
Outcomes by group	FactViolation + DimStudent	Join on FILE_ID, then to DimStudent
Sanctions by group	FactSanction + DimStudent	Join on FILE_ID, then to DimStudent

Important Considerations:

Consideration	Why It Matters
Comparison population	Are you comparing to enrolled population, residential population, or something else?
Small numbers	Disaggregated data for small groups may not be reliable and could risk identification
Intersectionality	A student's experience may be shaped by multiple identities simultaneously
Causation	Disparities are patterns to investigate, not conclusions about cause

Interpretation Guidance

For Decision-Makers: Equity analysis is essential but requires careful interpretation. A disparity is a *finding that demands investigation*, not proof of bias or discrimination. The data shows "what"—understanding "why" requires deeper inquiry into policies, practices, reporting patterns, and context.

For Analysts: DimStudent contains one row per unique student using the *most recent* record. If a student's classification changed (Freshman → Sophomore), DimStudent reflects their current status, not their status at the time of each incident. For true longitudinal demographic analysis at time of incident, you would need to preserve demographics per incident—a potential enhancement for future versions.

7.9 Repeat Involvement (Recidivism) Questions

These questions help you understand *how often students return* to the conduct system. This model provides two complementary approaches to measuring recidivism:

Approach	Table	Question It Answers
Within-year	FactRecidivism	"How many students had multiple cases this academic year?"
Cohort-based	FactCohortRecidivism	"Of students who first got in trouble in a given year, how many ever came back?"

Example Questions:

Within-Year (FactRecidivism):

- What percentage of students have multiple responsible findings in the same academic year?
- Is the within-year repeat rate increasing or decreasing over time?

Cohort-Based (FactCohortRecidivism):

- Of students whose first responsible finding was in AY2122, what percentage had any subsequent responsible finding?
- How quickly do students recidivate (days between first and second case)?
- Do certain student groups have higher long-term recidivism rates?
- Are certain interventions associated with lower long-term recurrence?
- How intense is repeat involvement (one repeat vs. three+ repeats)?

Where to Look:

Question Focus	Primary Table	Key Fields
Within-year repeat rates	FactRecidivism	Academic_Year, Found_Resp, Found_Resp_Again
Long-term recidivism by cohort	FactCohortRecidivism	Cohort_Year, Is_Recidivist
Time to recidivism	FactCohortRecidivism	Days_to_Recidivism
Intensity of repeat involvement	FactCohortRecidivism	Recidivism_Category, Repeat_Count

Question Focus	Primary Table	Key Fields
Recidivism by student group	FactCohortRecidivism + DimStudent	Join on SID

7.9.1 Understanding FactRecidivism (Within-Year):

Field	What It Contains
Academic_Year	The academic year being measured
Found_Resp	Count of unique students found responsible in at least one case
Found_Resp_Again	Count of unique students found responsible in more than one case
(Rate)	Found_Resp_Again ÷ Found_Resp (calculated in dashboard)

Definition: A student is a within-year recidivist if they were found responsible in **two or more separate cases** within the **same academic year**. Multiple responsible findings in a single case do not count as recidivism.

7.9.2 Understanding FactCohortRecidivism (Long-Term):

Field	What It Contains
SID	Student identifier (anonymized)
Cohort_Year	Academic year of student's first responsible finding
First_Incident_Date	Date of first responsible finding
Total_Responsible_Cases	Total cases with responsible findings (all time)
Repeat_Count	Number of cases after the first (0 = no recidivism)
Is_Recidivist	TRUE if student had any subsequent responsible finding
Recidivism_Category	Intensity: No Recidivism, One Repeat, Two Repeats, Three+ Repeats
Second_Incident_Date	Date of second responsible finding (NULL if none)
Days_to_Recidivism	Days between first and second case (NULL if none)

Definition: A student is assigned to a **cohort** based on the academic year of their **first-ever responsible finding**. They are a recidivist if they have **any subsequent case** with a responsible finding, regardless of when it occurs.

7.9.3 Comparing the Two Approaches:

Aspect	FactRecidivism	FactCohortRecidivism
Question answered	"How many repeat within a year?"	"How many ever come back?"
Time window	Single academic year	All time after first finding
Grain	One row per academic year	One row per student
Comparable to	Annual repeat rate	Criminal justice recidivism
Best for	Short-term patterns, annual reporting	Long-term trends, intervention evaluation

Interpretation Guidance

For Decision-Makers: Recidivism rates are one signal among many. A student returning to the conduct system isn't automatically a "failure"—it might mean they're still learning, they're in a challenging environment, or they have needs that conduct sanctions alone can't address. Use recidivism data to prompt questions about intervention effectiveness and student support.

When reviewing cohort-based recidivism, remember that **recent cohorts will always show lower rates** because less time has passed. The AY2425 cohort might show 5% recidivism while AY2122 shows 15%—but that's mostly because AY2122 students have had three more years to recidivate. Compare cohorts with similar follow-up time, or note the caveat when reporting.

Recidivism reflects continued system contact, not student intent, motivation, or responsiveness to support, and should not be interpreted as individual failure.

For Analysts: FactRecidivism provides aggregate counts by year—useful for dashboards but limited for deeper analysis. FactCohortRecidivism is at the student level, enabling joins to DimStudent for demographic disaggregation and to FactSanction

(via First_File_ID) for intervention analysis.

To analyze "do students who receive sanction X have lower recidivism than sanction Y?", join FactCohortRecidivism to FactSanction on the first case, then compare Is_Recidivist rates by sanction type. Note that this is observational—differences may reflect case severity, student characteristics, or other confounds, not sanction effectiveness.

7.10 Workload and Operational Questions

These questions help you understand *how work is distributed* and *where operational attention is needed*.

Example Questions:

- How many cases does each hearing officer handle per semester?
- Are case assignments balanced across staff, or are some officers overloaded?
- How many cases are currently pending? Overdue?
- Which case types consume the most staff time?

Where to Look:

Question Focus	Primary Table	Key Fields
Cases per officer	DimCase	ASSIGNED_TO, ASSIGNED_TO_NAME
Officer details	DimHearingOfficer	Department, Position
Pending/overdue cases	FactDeadline	DEADLINE, Overdue
Case volume by type	FactIncident or DimCase	TYPE

Interpretation Guidance

For Decision-Makers: Workload metrics support staffing conversations but don't capture everything. A hearing officer with fewer cases might handle more complex cases, supervise student staff, or have other responsibilities. Use workload data as a starting point for conversation, not a performance metric.

For Analysts: FactDeadline includes an `Overdue` flag calculated as `DEADLINE < today()`. This is calculated at the time the pipeline runs, so it reflects the state as of the last

> data refresh. For real-time monitoring, consider connecting Power BI directly to the source or scheduling frequent refreshes.

8 Data Transformation

This section contains the actual code that transforms raw Maxient exports into the star schema. Each subsection includes:

- A plain-language explanation of what's happening and why
- The R code that implements it
- A schema output showing the resulting table structure (where applicable)

If you're not reading the code: Focus on the explanations and schema outputs. They tell you what each step accomplishes without requiring you to understand the syntax.

If you are reading the code: The code is designed to be readable and maintainable. Related operations are grouped together, and comments explain non-obvious choices. Please note that this code is not directly transferable to other institutional contexts. Each institution configures it's datafeed to Maxient differently and that will impact how it needs to be cleaned and tidied.

8.1 Setup: Load Libraries and Custom Functions

Before any data transformation can happen, we need to load the tools that make it possible.

What This Does:

- Clears any previous data from memory (ensures a clean start)
- Loads publicly available R packages for data manipulation, Excel export, and secure hashing
- Loads custom functions written specifically for this pipeline

```
# Start with a clean environment
rm(list = ls())

# ——————————————————————————————————————————————————————————————————————
# Publicly Available Packages
# ——————————————————————————————————————————————————————————————————————
library(tidyverse)
library(writexl)
library(openssl)
library(digest)
```

```
# ──────────────────────────────────────────────────────────────────────────────
# Custom Functions and Configuration (included in appendices)
# ──────────────────────────────────────────────────────────────────────────────
source("Imports/conduct_funcs.R")
source("Imports/config.R")
```

Why This Matters

For Decision-Makers: This setup ensures that every time the pipeline runs, it starts fresh and uses the same tools. That's what makes results reproducible—you'll get the same output whether you run it today or next month.

For Analysts: The custom functions in `conduct_funcs.R` handle common operations (date conversion, anonymization, schema display). The configuration in `config.R` contains all mappings, date ranges, and constants. Separating these from the main pipeline makes maintenance easier—you can update a location mapping without touching the transformation logic.

8.2 Data Import

This section brings in all the raw data: multiple Maxient exports and the lookup tables that add institutional context.

8.2.1 Import Maxient Exports

Maxient exports are limited in size, so historical data spans multiple files. The pipeline reads each file and combines them into one dataset.

```
# ──────────────────────────────────────────────────────────────────────────────
# Load Raw Maxient Exports
# ──────────────────────────────────────────────────────────────────────────────

df1 <- read_csv(
  PATHS$imports$export1,
  show_col_types = FALSE
```

```
)

df2 <- read_csv(
  PATHS$imports$export2,
  show_col_types = FALSE
)

df3 <- read_csv(
  PATHS$imports$export3,
  show_col_types = FALSE,
  col_types = cols(
    .default = col_guess(),
    OTHER2 = col_character(),
    OTHER3 = col_character(),
    OTHER4 = col_character()
  )
)

# Combine all exports into one dataset
df <- bind_rows(df1, df2, df3)

# Free up memory by removing individual files
rm(df1, df2, df3)
```

8.2.2 Import Lookup Tables

Lookup tables add context that doesn't exist in Maxient: college names for major codes, staff names for assignment codes.

```
# ──────────────────────────────────────────────────────────────────────────────
# Lookup Tables
# ──────────────────────────────────────────────────────────────────────────────

# Academic Plan Codes → College names and major descriptions
apc <- read_csv(PATHS$imports$academic_plans, show_col_types = FALSE)
```

```
# Hearing Officer assignment codes → Staff names, departments, positions
ho <- read_csv(PATHS$imports$hearing_officers, show_col_types = FALSE)
```

Why Lookup Tables?

For Decision-Makers: Maxient stores major as a code (like "15BSBA"). That's not helpful for analysis. The lookup table translates it to "Business Administration" and tells us it's in the Lindner College of Business. This lets dashboards answer questions like "Which colleges have the highest involvement rates?"

For Analysts: Lookup tables are joined using `left_join()`. Unmatched values become NA, which we then replace with "Not Reported." If you see unexpected "Not Reported" values increasing, check whether the lookup tables need updating (new majors, new staff, etc.).

8.3 Tidy and Enrich Core Data

This section transforms the raw combined dataset into something usable for analysis. Each subsection addresses a specific data quality issue.

8.3.1 Standardize Locations and Join Context

What This Does:

- Converts location names to title case for consistency
- Applies the location mapping to standardize variations (e.g., "CALHOUN" → "Calhoun Hall")
- Filters to only residential locations (defined in `config.R`)
- Converts date columns from text to proper date format
- Joins lookup tables to add college and hearing officer context
- Replaces missing values with explicit "Not Reported" or "Unheard" labels

```
df <- df |>
  # ──────────────────────────────────────────────────────────────────────────

# Standardize location names
# ──────────────────────────────────────────────────────────────────────────
```

```
  mutate(
      INCIDENT_LOCATION = fix_nbsp(INCIDENT_LOCATION),
      INCIDENT_LOCATION = str_to_title(INCIDENT_LOCATION),
      INCIDENT_LOCATION = recode(INCIDENT_LOCATION, !!!LOCATION_MAP,
                                 .default = INCIDENT_LOCATION)
    ) |>
    # Keep only residential locations
    filter(INCIDENT_LOCATION %in% RED_LOCATIONS) |>

    # ──────────────────────────────────────────────────────────────────────────
    # Convert date columns
    # ──────────────────────────────────────────────────────────────────────────
    convert_to_date(c(
      "APPT_DATE", "CASE_CREATED_DATE", "DOB", "REPORTED_DATE",
      "INCIDENT_DATE", "HEARING_DATE", "DEADLINE"
    )) |>

    # ──────────────────────────────────────────────────────────────────────────
    # Join lookup tables for institutional context
    # ──────────────────────────────────────────────────────────────────────────
    left_join(apc, by = c("ACADEMIC_MAJOR" = "Academic Plan Code")) |>
    left_join(ho, by = c("ASSIGNED_TO" = "ASSIGNED_TO")) |>

    # ──────────────────────────────────────────────────────────────────────────
    # Rename columns for clarity
    # ──────────────────────────────────────────────────────────────────────────
    rename(
      Resolution = HEARING_TYPE,
      College    = `Academic Organization`,
      Major      = `Academic Plan`
    ) |>

    # ──────────────────────────────────────────────────────────────────────────
    # Replace NA values with explicit labels
    # ──────────────────────────────────────────────────────────────────────────
    mutate(
      Resolution       = replace_na(Resolution, "Unheard"),
      College          = replace_na(College, "Not Reported"),
      Major            = replace_na(Major, "Not Reported"),
```

```
    ASSIGNED_TO_NAME = replace_na(ASSIGNED_TO_NAME, "SCCS Staff"),
    CLASSIFICATION   = replace_na(CLASSIFICATION, "Not Reported")
  )
```

Why Explicit Missing Values?

For Decision-Makers: When a filter dropdown shows "Not Reported," you know there's data with missing information—and you can decide whether to include it. If we left these as blank, they'd silently disappear from many visualizations, potentially skewing results.

For Analysts: The `replace_na()` calls happen *after* the joins, catching both originally-missing values and values that failed to match in lookup tables. If "Not Reported" counts increase unexpectedly, investigate whether lookup tables are out of date.

8.3.2 Standardize Categorical Values

What This Does:

- Recodes classification values for consistency (e.g., "Pre-Junior" → "Sophomore")
- Standardizes athletics sport names (various abbreviations → consistent format)
- Standardizes role labels (e.g., "Alleged/Respondent" → "Respondent")

```
df <- df |>
  mutate(
    # ──────────────────────────────────────────────────────────────────────
    # Standardize classification
    # ──────────────────────────────────────────────────────────────────────
    CLASSIFICATION = recode(CLASSIFICATION,
                            "Pre-Junior" = "Sophomore",
                            "First Year (1L)" = "Graduate",
                            "Second Year (2L)" = "Graduate",
                            "Third Year (3L)" = "Graduate",
                            "Student" = "Not Reported"),

    # ──────────────────────────────────────────────────────────────────────
    # Standardize athletics sport names
    # ──────────────────────────────────────────────────────────────────────
```

```
    MEM_ATHLETICS_SPORT = recode(
      MEM_ATHLETICS_SPORT,
      "Men's Track &amp; Field"          = "Men's Track and Field",
      "Women's Track &amp; Field"        = "Women's Track and Field",
      "Women's Indoor Track &amp; Field" = "Women's Track and Field",
      "M-Football"                        = "Men's Football",
      "M-Baseball"                        = "Men's Baseball",
      "Baseball"                          = "Men's Baseball",
      "Cheerleaders"                      = "Cheer",
      "Football"                          = "Men's Football",
      "Manager"                           = "Athletic Manager",
      "Men's Swimming &amp; Diving"      = "Men's Swim and Dive",
      "N/A"                               = "Not Athlete",
      "Women's Swimming &amp; Diving"    = "Women's Swim and Dive",
      "Trainer"                           = "Athletic Trainer"
    ),

    # ──────────────────────────────────────────────────────────────────────
    # Standardize role labels
    # ──────────────────────────────────────────────────────────────────────
    ROLE = recode(ROLE, "Alleged/Respondent" = "Respondent")
  ) |>
  filter(CLASSIFICATION %nin% c('Staff', 'Student Organization'))
```

8.3.3 Calculate Timeline Metrics

What This Does:

Calculates three measures of how long cases take to move through the process:

Metric	Measures
`reported_to_case`	Days from incident report to case creation
`case_to_resolution`	Days from case creation to hearing/resolution
`reported_to_resolution`	Total days from report to resolution

The calculations subtract institutional pause periods (breaks, holidays) and prevent negative values from data entry errors.

```r
df <- df |>
  mutate(
    # ──────────────────────────────────────────────────────────────────────
    # Calculate raw day differences, then subtract pause periods
    # ──────────────────────────────────────────────────────────────────────
    reported_to_case =
      as.numeric(difftime(CASE_CREATED_DATE, REPORTED_DATE,
                          units = "days")) -
      compute_overlaps(REPORTED_DATE, CASE_CREATED_DATE, Resolution,
                       OMIT_RANGES),

    case_to_resolution =
      as.numeric(difftime(HEARING_DATE, CASE_CREATED_DATE,
                          units = "days")) -
      compute_overlaps(CASE_CREATED_DATE, HEARING_DATE, Resolution,
                       OMIT_RANGES),

    reported_to_resolution =
      as.numeric(difftime(HEARING_DATE, REPORTED_DATE,
                          units = "days")) -
      compute_overlaps(REPORTED_DATE, HEARING_DATE, Resolution,
                       OMIT_RANGES)
  ) |>

  # ──────────────────────────────────────────────────────────────────────
  # Floor negative values to zero (handles data entry errors)
  # ──────────────────────────────────────────────────────────────────────
  mutate(
    across(
      c(reported_to_case, case_to_resolution, reported_to_resolution),
      ~ pmax(.x, 0)
    )
  )
```

Why Adjust for Pause Periods?

For Decision-Makers: Without this adjustment, a case reported December 15 and resolved January 15 would show as 31 days—even though the office was closed for 20 of those days. Subtracting pause periods gives a more accurate picture of *working*

time in the system.

For Analysts: Pause periods are defined in `config.R` as `OMIT_RANGES`. The `compute_overlaps()` function (from `conduct_funcs.R`) calculates how many days of each case's timeline fall within these ranges. Warning Letter cases skip adjustment because they resolve immediately. If you add new pause periods (like a COVID closure), add them to `OMIT_RANGES`.

8.3.4 Filter Population and Add Academic Time Fields

What This Does:

- Removes non-student records (locations, non-students, "Other" roles)
- Creates academic year based on incident date
- Adds time-based fields for trend analysis (week, weekday, month)
- Calculates age at incident
- Renames UC-specific "OTHER" fields to meaningful names
- Standardizes college names to abbreviations

```
df <- df |>
  # ──────────────────────────────────────────────────────────────────────────
  # Filter to student respondents only
  # ──────────────────────────────────────────────────────────────────────────
  filter(!ROLE %in% c("Location", "Non-Student", "Non-student",
                      "Other")) |>
  filter(CLASSIFICATION != "Non-Student") |>

  # ──────────────────────────────────────────────────────────────────────────
  # Create academic year and time-based fields
  # ──────────────────────────────────────────────────────────────────────────
  mutate(
    ACADEMIC_YEAR = case_when(
      INCIDENT_DATE >= ymd("2017-08-01") &
        INCIDENT_DATE <= ymd("2018-07-31") ~ "AY1718",
      INCIDENT_DATE >= ymd("2018-08-01") &
        INCIDENT_DATE <= ymd("2019-07-31") ~ "AY1819",
      INCIDENT_DATE >= ymd("2019-08-01") &
```

```
      INCIDENT_DATE <= ymd("2020-07-31") ~ "AY1920",
    INCIDENT_DATE >= ymd("2020-08-01") &
      INCIDENT_DATE <= ymd("2021-07-31") ~ "AY2021",
    INCIDENT_DATE >= ymd("2021-08-01") &
      INCIDENT_DATE <= ymd("2022-07-31") ~ "AY2122",
    INCIDENT_DATE >= ymd("2022-08-01") &
      INCIDENT_DATE <= ymd("2023-07-31") ~ "AY2223",
    INCIDENT_DATE >= ymd("2023-08-01") &
      INCIDENT_DATE <= ymd("2024-07-31") ~ "AY2324",
    INCIDENT_DATE >= ymd("2024-08-01") &
      INCIDENT_DATE <= ymd("2025-07-31") ~ "AY2425",
    TRUE ~ "AY2526"
  ),

  # Week number for week-over-week analysis
  WEEK_NUM = isoweek(INCIDENT_DATE),

  # Day of week for day patterns
  incident_wday = wday(INCIDENT_DATE, label = TRUE),

  # Month ordered by academic year (Aug-Jul) rather than calendar year
  incident_month = factor(
    month(INCIDENT_DATE, label = TRUE, abbr = TRUE),
    levels = c("Aug", "Sep", "Oct", "Nov", "Dec",
               "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul")
  ),

  # Age at time of incident
  age_at_incident = round((DOB %--% INCIDENT_DATE) / years(1), 2)
) |>

# ──────────────────────────────────────────────────────────────────────────
# Rename UC-specific fields
# ──────────────────────────────────────────────────────────────────────────
rename(
  Regional_Campus = OTHER2,
  College2        = OTHER3,
  Citizenship     = OTHER4
)
```

```
# ------------------------------------------------------------------------------
# Standardize college names to abbreviations
# ------------------------------------------------------------------------------
df <- df |>
  mutate(College = recode(College, !!!COLLEGE_REPLACEMENTS,
                          .default = College))
```

Why Filter to Students Only?

For Decision-Makers: The conduct dashboards are designed to analyze *student* conduct. Including "Location" records (used for tracking where something happened without a person attached) or non-student records would inflate counts and create confusing results.

For Analysts: The `case_when()` for ACADEMIC_YEAR is explicit rather than using `compute_academic_year()` from conduct_funcs.R to handle edge cases with dates outside the expected range. If you extend the data range, add new conditions or update the default case.

8.3.5 Normalize Ethnicity Values

What This Does: Standardizes the many variations of ethnicity codes in the source data into consistent reporting categories.

```
df <- df |>
  mutate(
    # ------------------------------------------------------------------------------
    # Preserve raw value, then normalize
    # ------------------------------------------------------------------------------
    ETHNICITY_raw = ETHNICITY,
    ETHNICITY = ETHNICITY |>
      str_trim() |>
      str_replace_all("[,/]+", " ") |>
      str_squish() |>
      toupper()
  ) |>
  mutate(
```

```
    # ──────────────────────────────────────────────────────────────────────────
    # Map to standard categories
    # ──────────────────────────────────────────────────────────────────────────
    ETHNICITY_BUCKET = case_when(
      ETHNICITY %in% c("HISPA") ~ "Hispanic",
      ETHNICITY %in% c("AMIND") ~ "American Indian",
      ETHNICITY %in% c("NHOPO") ~ "Native Hawaiian or Pacific Islander",
      ETHNICITY %in% c("BLACK", "BLACK NOTHISP", "AFRICAN-AMERICAN")
      ~ "Black",
      ETHNICITY %in% c("WHITE", "NOTHISP WHITE", "CAUCASIAN") ~ "White",
      ETHNICITY %in% c("ASIAN", "ASIAN NOTHISP") ~ "Asian",
      ETHNICITY %in% c("NSPEC") ~ "Not Reported",
      ETHNICITY %in% c("NOTHISP", "OTHER", "OTHER/NO RESPONSE") ~ "Other",

      # Multi-word codes (after normalization) indicate multi-ethnic
      str_detect(ETHNICITY, "\\b[A-Z]+\\b\\s+\\b[A-Z]+\\b")
      ~ "Multi-Ethnic",

      # Fallback cases
      is.na(ETHNICITY) ~ "Not Reported",
      TRUE ~ "Other"
    ),

    # ──────────────────────────────────────────────────────────────────────────
    # Convert to ordered factor
    # ──────────────────────────────────────────────────────────────────────────
    ETHNICITY = factor(
      ETHNICITY_BUCKET,
      levels = c("Hispanic", "American Indian",
                 "Native Hawaiian or Pacific Islander",
                 "Black", "White", "Asian", "Multi-Ethnic",
                 "Other", "Not Reported")
    )
  ) |>
  select(-ETHNICITY_BUCKET)
```

Why Standardize Ethnicity?

For Decision-Makers: Equity analysis requires consistent categories. If "Black" students appear as "BLACK," "Black NotHisp," and "African-American" in different records, they'll be split across three categories in any chart—hiding the actual pattern.

For Analysts: The raw value is preserved in `ETHNICITY_raw` for troubleshooting. If new codes appear in future exports, they'll fall into "Other"—monitor for unexpected growth in that category. The factor levels control sort order in visualizations.

8.3.6 Anonymize Identifiable Fields

What This Does: Replaces real identifiers (case numbers, file IDs, student IDs) with scrambled codes that cannot be reversed to reveal the original values.

```
# ──────────────────────────────────────────────────────────────────────────────
# Load the encryption key
# ──────────────────────────────────────────────────────────────────────────────
pepper <- readBin(PATHS$pepper, "raw", 32)

# ──────────────────────────────────────────────────────────────────────────────
# Apply hashing to identifier columns
# ──────────────────────────────────────────────────────────────────────────────
df <- df |>
  mutate(
    CASE_NUMBER = hash_col(CASE_NUMBER, pepper),
    FILE_ID     = hash_col(FILE_ID, pepper),
    SID         = hash_col(SID, pepper)
  )
```

How Anonymization Works

For Decision-Makers: After this step, a student ID like "M12345678" becomes something like "a3f7c9e2b1d8...". The scrambled code is consistent—the same student always gets the same code—so you can still track repeat involvement and link records across tables. But no one can work backward from the code to find the real ID.

For Analysts: The `pepper.bin` file is the encryption key. It must be:

- **Protected**: Don't commit to version control or share via email
- **Backed up**: If lost, you can't regenerate consistent hashes
- **Consistent**: Use the same key for every pipeline run, or hashes won't match across refreshes

The `hash_col()` function in `conduct_funcs.R` uses HMAC-SHA256 and returns the first 32 hex characters.

8.3.7 Prepare Sanctions for Extraction

What This Does: Creates a separate dataset of sanctions that will be used later to build the FactSanction table.

```
# ──────────────────────────────────────────────────────────────────────────────
# Extract sanction columns for later use
# ──────────────────────────────────────────────────────────────────────────────
sanctions <- df |>
  select(FILE_ID, INCIDENT_DATE, ADDENDUM:SUSPENSION)
```

8.3.8 Review: Structure After Cleaning

This diagnostic output shows the structure of the cleaned dataset before it's split into dimension and fact tables.

Reading the output:

- Lines 1–37: Standard Maxient fields
- Lines 38–42: UC-specific tag fields
- Lines 43–72: Sanction columns (UC-specific)
- Lines 73–75: UC "OTHER" fields (renamed above)
- Lines 76+: Fields added through lookups and calculations

```r
df_schema(df) |>
  select(column, type) |>
  print(n = Inf)
```

```
# A tibble: 89 x 2
   column                                                type
   <chr>                                                 <chr>
 1 FILE_ID                                               character
 2 APPT_DATE                                             Date
 3 CASE_NUMBER                                           character
 4 DEADLINE_REASON                                       character
 5 DEADLINE                                              Date
 6 SID                                                   character
 7 ASSIGNED_TO                                           character
 8 STATUS                                                character
 9 TYPE                                                  character
10 CASE_CREATED_DATE                                     Date
11 HOLD_IN_PLACE                                         character
12 GENDER                                                character
13 ACADEMIC_MAJOR                                        character
14 GPA_CUME                                              numeric
15 CLASSIFICATION                                        character
16 DOB                                                   Date
17 MEM_ATHLETICS_SPORT                                   character
18 ETHNICITY                                             factor
19 REPORTED_DATE                                         Date
20 CLERY_REPORTABILITY                                   character
21 INCIDENT_LOCATION                                     character
22 ROLE                                                  character
23 INCIDENT_DATE                                         Date
24 CHARGE_1                                              character
25 FINDING_1                                             character
26 CHARGE_2                                              character
27 FINDING_2                                             character
28 HEARING_DATE                                          Date
29 CHARGE_3                                              character
30 FINDING_3                                             character
31 Resolution                                            character
32 CHARGE_4                                              character
33 FINDING_4                                             character
```

```
34 CHARGE_5                                                 character
35 FINDING_5                                                character
36 CHARGE_6                                                 character
37 FINDING_6                                                character
38 Tag: Case Dismissed by SCCS Determination                character
39 Tag: Case Dismissed/Complainant Non-Compliant            character
40 Tag: Hospital Transport                                  character
41 Tag: Report Dismissed/Complainant Form Not Received      character
42 Tag: Residence Halls Vandalism                           character
43 ADDENDUM                                                 character
44 ALDRPAPER                                                character
45 ANGERMGMT                                                character
46 APOLOGY                                                  character
47 ASEP                                                     character
48 BASICS                                                   character
49 BEHAVIORMOD                                              character
50 BULLETIN                                                 character
51 CANNABISCRS                                              character
52 COMMUNITY_SERVICE                                        character
53 DMS                                                      character
54 EDU_SIGN                                                 character
55 EXPULSION                                                character
56 FIRESAFEED                                               character
57 INTERIMSUSP                                              character
58 LOSSPRIV                                                 character
59 NOCONTACT                                                character
60 PARENTNOTIFY                                             character
61 PFLDRUG                                                  character
62 REDEXPUL                                                 character
63 REDSUSP                                                  character
64 REDTRANSFER                                              character
65 REFLECTCONT                                              character
66 REFLGTUL                                                 character
67 REPRIMAND                                                character
68 RESTITUTION                                              character
69 RESTRICTION                                              character
70 THEFTAWARE                                               character
71 DP                                                       character
72 SUSPENSION                                               character
73 Citizenship                                              character
```

```
74 Regional_Campus                    character
75 College2                           character
76 Major                              character
77 College                            character
78 ASSIGNED_TO_NAME                   character
79 OFFICE                             character
80 POSITION                           character
81 reported_to_case                   numeric
82 case_to_resolution                 numeric
83 reported_to_resolution             numeric
84 ACADEMIC_YEAR                      character
85 WEEK_NUM                           numeric
86 incident_wday                      ordered, factor
87 incident_month                     ordered, factor
88 age_at_incident                    numeric
89 ETHNICITY_raw                      character
```

8.4 Star Schema Tables

This section constructs the final tables that Power BI will use. Each table has a specific purpose and a clear grain (what one row represents).

Quick Reference:

Table Type	Purpose	Examples
Dimension	Descriptive context for filtering	DimDate, DimStudent, DimHousing
Fact	Events and measures to count/analyze	FactIncident, FactViolation, FactSanction

8.4.1 Dimensions

Dimension tables store descriptive context that's reused across multiple fact tables. They enable consistent filtering and prevent repeating the same information in every row of the fact tables.

8.4.1.1 DimCase

What It Contains: One row per unique case, with stable case-level attributes.

Use This When: You need to filter or analyze by case characteristics (type, status, assignment) rather than individual incidents or violations.

Key Fields	Description
FILE_ID	Unique case identifier (anonymized)
CASE_NUMBER	Case number (anonymized)
TYPE	Case type (Residential, Academic, etc.)
STATUS	Current status (Closed, Open, etc.)
Resolution	How the case was resolved
ASSIGNED_TO	Hearing officer assignment code
ASSIGNED_TO_NAME	Hearing officer name

```r
DimCase <- df |>
  select(
    FILE_ID,
    CASE_NUMBER,
    TYPE,
    STATUS,
    HOLD_IN_PLACE,
    ASSIGNED_TO,
    ASSIGNED_TO_NAME,
    Resolution,
    CASE_CREATED_DATE
  ) |>
  mutate(
    HOLD_IN_PLACE = case_when(
      HOLD_IN_PLACE == "Yes" ~ TRUE,
      TRUE ~ FALSE
    )
  ) |>
  distinct()

df_schema(DimCase) |>
  select(column, type, example) |>
  mutate(across(3, ~ replace(.x, 6, "Hidden"))) |>
  mutate(across(3, ~ replace(.x, 7, "Hidden"))) |>
  print(n = Inf)
```

```
# A tibble: 9 x 3
  column            type      example
  <chr>             <chr>     <chr>
1 FILE_ID           character 42ee9d11f4bf4611516f7736b5ec4aca
2 CASE_NUMBER       character c21769dbf6688f856c30587d2e10fd71
3 TYPE              character Non Academic Misconduct
4 STATUS            character Closed
5 HOLD_IN_PLACE     logical   FALSE
6 ASSIGNED_TO       character Hidden
7 ASSIGNED_TO_NAME  character Hidden
8 Resolution        character Procedural Review
9 CASE_CREATED_DATE Date      2018-09-25
```

When to Use DimCase

For Decision-Makers: Use case-level analysis when your question is about workload, staffing, or case characteristics. "How many cases did Officer Smith handle?" is a case-level question.

For Analysts: `distinct()` ensures one row per case. If a case has multiple incidents or students, it still appears once here. Join to fact tables on FILE_ID.

8.4.1.2 DimCollege

What It Contains: A controlled list of academic colleges referenced in the data.

```
DimCollege <- df |>
  distinct(College) |>
  arrange(College)

df_schema(DimCollege) |>
  select(column, type, example) |>
  print(n = Inf)
```

```
# A tibble: 1 x 3
  column  type      example
  <chr>   <chr>     <chr>
1 College character ALC
```

8.4.1.3 DimDate

What It Contains: One row per calendar date, spanning the full analysis period. Includes both calendar fields (year, month, day) and academic time fields (academic year, semester, term).

Why This Matters: This is the backbone of all time-based analysis. Every fact table connects to DimDate through a date field, enabling consistent filtering by semester, academic year, or any other time period.

Key Fields	Description
Date	The calendar date
Academic Year	AY format (e.g., “AY2324”)
Semester	FA, SP, or SU
AcademicTermKey	Numeric key for sorting terms correctly
AY_Order	Numeric key for sorting academic years

```
# ──────────────────────────────────────────────────────────────────────────────
# Generate complete date spine
# ──────────────────────────────────────────────────────────────────────────────

DimDate <- tibble(
  Date = seq.Date(DATE_RANGE$start, DATE_RANGE$end, by = "day")
) |>

  # ──────────────────────────────────────────────────────────────────────────────
  # Standard calendar fields
  # ──────────────────────────────────────────────────────────────────────────────
  mutate(
    Year         = year(Date),
    Month        = month(Date),
    `Month Name` = month(Date, label = TRUE, abbr = FALSE),
    Day          = day(Date),
    `ISO Week`   = isoweek(Date),
    Weekday      = wday(Date, label = TRUE, abbr = FALSE)
  ) |>

  # ──────────────────────────────────────────────────────────────────────────────
  # Academic year (Aug 1 – Jul 31)
  # ──────────────────────────────────────────────────────────────────────────────
  mutate(
    `Academic Year` = if_else(
      Date >= make_date(year(Date), 8, 1),
      paste0("AY", year(Date) - 2000, year(Date) - 1999),
      paste0("AY", year(Date) - 2001, year(Date) - 2000)
```

```
  )
) |>

# ──────────────────────────────────────────────────────────────────────────────
# Sort keys for correct ordering in visuals
# ──────────────────────────────────────────────────────────────────────────────
mutate(
  AY_Order = as.integer(substr(`Academic Year`, 3, 6))
) |>

# ──────────────────────────────────────────────────────────────────────────────
# Semester classification
# ──────────────────────────────────────────────────────────────────────────────
mutate(
  Semester = case_when(
    Month >= 8 & Month <= 12 ~ "FA",
    Month >= 1 & Month <= 4  ~ "SP",
    TRUE                     ~ "SU"
  )
) |>

# ──────────────────────────────────────────────────────────────────────────────
# Academic term key (enables correct term sorting)
# Format: [2-digit AY start] * 10 + [term number]
# Example: AY2324 Fall = 23 * 10 + 1 = 231
# ──────────────────────────────────────────────────────────────────────────────
mutate(
  AcademicTermKey = case_when(
    Semester == "FA" ~ as.integer(
      substr(`Academic Year`, 3, 4)) * 10 + 1L,
    Semester == "SP" ~ as.integer(
      substr(`Academic Year`, 3, 4)) * 10 + 2L,
    Semester == "SU" ~ as.integer(
      substr(`Academic Year`, 3, 4)) * 10 + 3L,
    TRUE             ~ NA_integer_
  )
) |>
arrange(Date) |>
# ──────────────────────────────────────────────────────────────────────────────
```

```
    # Sort helpers for visuals
    # ──────────────────────────────────────────────────────────────────────────
    mutate(
      Month_Sort = case_when(
        Month == 8  ~ 1L,
        Month == 9  ~ 2L,
        Month == 10 ~ 3L,
        Month == 11 ~ 4L,
        Month == 12 ~ 5L,
        Month == 1  ~ 6L,
        Month == 2  ~ 7L,
        Month == 3  ~ 8L,
        Month == 4  ~ 9L,
        Month == 5  ~ 10L,
        Month == 6  ~ 11L,
        Month == 7  ~ 12L
      ),
      Weekday_Sort = wday(Date, week_start = 7)  # Sunday = 1
    )

  df_schema(DimDate) |>
    select(column, type, example) |>
    print(n = Inf)
```

```
# A tibble: 13 x 3
   column          type            example
   <chr>           <chr>           <chr>
 1 Date            Date            2019-08-01
 2 Year            numeric         2019
 3 Month           numeric         8
 4 Month Name      ordered, factor August
 5 Day             integer         1
 6 ISO Week        numeric         31
 7 Weekday         ordered, factor Thursday
 8 Academic Year   character       AY1920
 9 AY_Order        integer         1920
10 Semester        character       FA
11 AcademicTermKey numeric         191
12 Month_Sort      integer         1
13 Weekday_Sort    numeric         5
```

Why a Separate Date Table?

For Decision-Makers: A dedicated date table ensures that "Fall 2023" means exactly the same thing on every page of every dashboard. Without it, different visuals might calculate time periods differently, leading to confusion.

For Analysts: In PowerBI, mark this as the date table and create relationships from fact tables' date columns to DimDate[Date]. The AcademicTermKey and AY_Order fields enable correct sorting—without them, "AY2324" would sort before "AY1920" alphabetically.

8.4.1.4 DimAcademicTerm

What It Contains: One row per academic term (Fall, Spring, Summer for each academic year). Provides labels and sort keys for term-level analysis.

```
DimAcademicTerm <- DimDate |>
  filter(Semester %in% c("FA", "SP", "SU")) |>
  distinct(
    Academic_Year = `Academic Year`,
    Semester,
    AcademicTermKey
  ) |>
  mutate(
    AcademicTermLabel = paste(Academic_Year, Semester),
    TermSort          = AcademicTermKey
  ) |>
  arrange(TermSort)

df_schema(DimAcademicTerm) |>
  select(column, type, example) |>
  print(n = Inf)
```

```
# A tibble: 5 x 3
  column        type      example
  <chr>         <chr>     <chr>
1 Academic_Year character AY1920
```

```
2 Semester          character FA
3 AcademicTermKey   numeric   191
4 AcademicTermLabel character AY1920 FA
5 TermSort          numeric   191
```

8.4.1.5 DimAcademicYear

What It Contains: One row per academic year with explicit start/end dates and sort order.

```
DimAcademicYear <- DimDate |>
  distinct(`Academic Year`) |>
  rename(ACADEMIC_YEAR = `Academic Year`) |>
  mutate(
    AY_Start_Year = as.integer(paste0("20", str_sub(ACADEMIC_YEAR, 3, 4))),
    AY_Start_Date = as.Date(paste0(AY_Start_Year, "-08-01")),
    AY_End_Date   = as.Date(paste0(AY_Start_Year + 1, "-07-31")),
    AY_Order      = as.integer(str_sub(ACADEMIC_YEAR, 3, 6))
  ) |>
  select(ACADEMIC_YEAR, AY_Order, AY_Start_Date, AY_End_Date) |>
  arrange(AY_Order)

df_schema(DimAcademicYear) |>
  select(column, type, example) |>
  print(n = Inf)
```

```
# A tibble: 4 x 3
  column        type      example
  <chr>         <chr>     <chr>
1 ACADEMIC_YEAR character AY1920
2 AY_Order      integer   1920
3 AY_Start_Date Date      2019-08-01
4 AY_End_Date   Date      2020-07-31
```

> *Why Explicit Year Boundaries?*
>
> **For Decision-Makers:** When someone asks "What happened in AY2324?", there's no ambiguity: August 1, 2023 through July 31, 2024. These explicit boundaries prevent

the “off by a month” errors that plague calendar-year reporting.

For Analysts: Use AY_Order for sort order in visuals. The start/end dates can be used for date range filtering or for calculating whether a date falls within a specific academic year.

8.4.1.6 DimHearingOfficer

What It Contains: Staff metadata for workload and assignment analysis.

```
DimHearingOfficer <- ho |>
  distinct()

df_schema(DimHearingOfficer) |>
  select(column, type, example) |>
  mutate(across(3, ~ replace(.x, 1, "Hidden"))) |>
  mutate(across(3, ~ replace(.x, 2, "Hidden"))) |>
  print(n = Inf)
```

```
# A tibble: 4 x 3
  column           type      example
  <chr>            <chr>     <chr>
1 ASSIGNED_TO      character Hidden
2 ASSIGNED_TO_NAME character Hidden
3 OFFICE           character RED
4 POSITION         character Area Coordinator
```

8.4.1.7 DimHousing

What It Contains: One row per residence hall with stable building attributes (type, ownership, room style). This data is maintained in partnership with University Housing.

```
DimHousing <- read_csv(
  PATHS$imports$dim_housing,
  locale = locale(encoding = "Windows-1252")
```

```
) |>
  mutate(INCIDENT_LOCATION = fix_nbsp(INCIDENT_LOCATION)) |>
  arrange(INCIDENT_LOCATION)

df_schema(DimHousing) |>
  select(column, type, example) |>
  mutate(across(3, ~ replace(.x, 1, "Hidden"))) |>
  print(n = Inf)
```

```
# A tibble: 4 x 3
  column            type      example
  <chr>             <chr>     <chr>
1 INCIDENT_LOCATION character Hidden
2 housing_type      character Apartment
3 ownership         character University-Owned
4 room_style        character Apartments
```

8.4.1.8 DimHousingYear

What It Contains: Housing attributes that can change from year to year (capacity, population type, etc.).

Why Separate from DimHousing? A building's capacity or use might change. Separating year-specific attributes preserves historical context—you can see what a building was like in AY2122, not just what it's like now.

```
DimHousingYear <- read_csv(
  PATHS$imports$dim_housing_year,
  locale = locale(encoding = "Windows-1252")
) |>
  mutate(INCIDENT_LOCATION = fix_nbsp(INCIDENT_LOCATION)) |>
  arrange(INCIDENT_LOCATION)

df_schema(DimHousingYear) |>
  select(column, type, example) |>
  mutate(across(3, ~ replace(.x, 1, "Hidden"))) |>
  print(n = Inf)
```

```
# A tibble: 9 x 3
  column            type      example
  <chr>             <chr>     <chr>
1 INCIDENT_LOCATION character Hidden
2 ACADEMIC_YEAR     character AY2526
3 RESIDENT_TYPE     character Mixed
4 CAPACITY_MAX      logical   <NA>
5 N_RAS             logical   <NA>
6 HAS_FRONT_DESK    logical   FALSE
7 IS_HOTEL          logical   FALSE
8 IS_ACTIVE         logical   TRUE
9 NOTES             logical   <NA>
```

8.4.1.9 DimSanction

What It Contains: Sanction metadata including categories, severity levels, and flags for analysis.

Key Fields	Description
Sanction	The sanction code
CATEGORY_LABEL	Purpose category (Educational, Status, Restrictive, etc.)
RESTRICTIVE_FLAG	Whether the sanction restricts student behavior/access
SEVERITY_LEVEL	Numeric severity (0=Educational to 4=Removal)
SEVERITY_LABEL	Text label for severity level

Severity Scale:

Level	Label	Examples
0	Educational	Online modules, reflection papers, etc.
1	Status (Non-Restrictive)	Probation, reprimand
2	Conditional Restriction	Probation, deferred suspension
3	Restrictive	No-contact orders, loss of privileges

Level	Label	Examples
4	Removal	Suspension, housing termination

```
DimSanction <- read_csv(PATHS$imports$dim_sanction)

df_schema(DimSanction) |>
  select(column, type, example) |>
  print(n = Inf)
```

```
# A tibble: 7 x 3
  column            type      example
  <chr>             <chr>     <chr>
1 SANCTION          character ACDISMISS
2 DESCRIPTIVE_NAME  character Academic Dismissal
3 SANCTION_CATEGORY character Academic
4 CATEGORY_LABEL    character Academic
5 RESTRICTIVE_FLAG  logical   TRUE
6 SEVERITY_LEVEL    numeric   4
7 SEVERITY_LABEL    character Removal
```

Why Categorize Sanctions?

For Decision-Makers: Knowing that we issued 500 sanctions isn't as useful as knowing we issued 350 educational interventions, 100 status sanctions, and 50 restrictive sanctions. Categories help you understand your intervention philosophy in practice.

For Analysts: Join FactSanction to DimSanction on the Sanction field. You can then group by CATEGORY_LABEL or SEVERITY_LEVEL for aggregate analysis. The RESTRICTIVE_FLAG enables quick filtering to "sanctions that limit student access."

8.4.1.10 DimStudent

What It Contains: One row per unique student using their most recent record. Includes demographic and academic context for equity analysis. Because this uses the most recent snapshot, a student's attributes (like classification) reflect their current status, not their status at the time of each incident.

```
DimStudent <- df |>
  filter(!is.na(SID)) |>
  arrange(SID, INCIDENT_DATE) |>
  group_by(SID) |>
  slice_tail(n = 1) |>
  ungroup() |>
  select(
    SID,
    DOB,
    GENDER,
    ETHNICITY,
    CLASSIFICATION,
    MEM_ATHLETICS_SPORT,
    GPA_CUME,
    ACADEMIC_MAJOR,
    Major,
    Regional_Campus,
    College2,
    Citizenship
  )

df_schema(DimStudent) |>
  select(column, type, example) |>
  mutate(across(3, ~ replace(.x, 2, NA))) |>
  mutate(across(3, ~ replace(.x, 10, "Hidden"))) |>
  mutate(across(3, ~ replace(.x, 11, "Hidden"))) |>
  print(n = Inf)
```

```
# A tibble: 12 x 3
   column              type      example
   <chr>               <chr>     <chr>
 1 SID                 character 000200be597548473fa89a95000dbdb8
```

```
 2 DOB                 Date      <NA>
 3 GENDER              character Male
 4 ETHNICITY           factor    White
 5 CLASSIFICATION      character Sophomore
 6 MEM_ATHLETICS_SPORT character Not Athlete
 7 GPA_CUME            numeric   3.208
 8 ACADEMIC_MAJOR      character RE-BBA
 9 Major               character Real Estate
10 Regional_Campus     character Hidden
11 College2            character Hidden
12 Citizenship         character Domestic
```

Privacy and Purpose

For Decision-Makers: This table supports pattern analysis (e.g., "Are any student groups over-represented?"), not individual case lookup. Student IDs are anonymized, so you can see patterns without identifying specific students.

For Analysts: The `slice_tail(n = 1)` after sorting by date ensures we keep the most recent record per student. If you need demographics *at time of incident* for longitudinal demographic analysis, you'd need to preserve that in the fact tables—a potential future enhancement.

8.4.1.11 DimViolation

What It Contains:: Charge metadata including categories, severity levels, and flags for analysis.

Field	Description
VIOLATION	The violation/charge name
POLICY	Policy area (Academic Integrity, Alcohol, Drugs, Harassment, Residential, General Conduct, etc.)
CATEGORY	Behavior category (Substance Use, Interpersonal Harm, Property, Safety, Administrative, etc.)
IS_ALCOHOL	Whether the violation involves alcohol
IS_DRUG	Whether the violation involves drugs/narcotics
IS_VIOLENCE	Whether the violation involves violence or threat of harm

Field	Description
SEVERITY_LEVEL	Numeric severity (1–5)
SEVERITY_LABEL	Text label for severity level
RATIONALE	Brief explanation of severity classification

Severity Scale:

Level	Label	Examples
1	Minor	Quiet hours, guest policy, tobacco, passive participation
2	Moderate	Underage alcohol, paraphernalia, failure to comply, dishonesty
3	Serious	Cheating, drug possession, theft, fire safety, destruction of property
4	Severe	Hazing, harassment, weapons, public endangerment, retaliation
5	Critical	Distribution, sexual misconduct, stalking, physical violence, domestic violence

```
DimViolation <- read_csv(
  PATHS$imports$dim_charge)

df_schema(DimViolation) |>
  select(column, type, example) |>
  print(n = Inf)
```

```
# A tibble: 9 x 3
  column         type      example
  <chr>          <chr>     <chr>
1 VIOLATION      character Academic Misconduct - Aiding and Abetting
2 POLICY         character Student Code of Conduct
3 CATEGORY       character Academic Misconduct
```

```
4 IS_ALCOHOL     logical   FALSE
5 IS_DRUG        logical   FALSE
6 IS_VIOLENCE    logical   FALSE
7 SEVERITY_LEVEL numeric   2
8 SEVERITY_LABEL character Moderate
9 RATIONALE      character Facilitation, not primary actor
```

8.4.2 Fact Tables

Fact tables store events, outcomes, and measures. Each table has a clear grain—what one row represents—which makes counts and calculations reliable.

8.4.2.1 FactDeadline

Grain: One row per case with a deadline set.

What It Contains: Case deadlines and whether they're overdue. Supports operational monitoring.

```
FactDeadline <- df |>
  filter(!is.na(DEADLINE)) |>
  mutate(
    Overdue = DEADLINE < today()
  ) |>
  select(
    FILE_ID,
    TYPE,
    STATUS,
    DEADLINE,
    DEADLINE_REASON,
    HOLD_IN_PLACE,
    ASSIGNED_TO,
    ACADEMIC_YEAR,
    Overdue
  )

df_schema(FactDeadline) |>
```

```
    select(column, type, example) |>
    mutate(across(3, ~ replace(.x, 7, "Hidden"))) |>
    print(n = Inf)
```

```
# A tibble: 9 x 3
  column          type      example
  <chr>           <chr>     <chr>
1 FILE_ID         character a02d6733d689485726c811c868f7dfb1
2 TYPE            character Non Academic Misconduct
3 STATUS          character Closed
4 DEADLINE        Date      9999-12-31
5 DEADLINE_REASON character Loss of Privileges in Effect
6 HOLD_IN_PLACE   character Yes
7 ASSIGNED_TO     character Hidden
8 ACADEMIC_YEAR   character AY1920
9 Overdue         logical   FALSE
```

Real-Time Monitoring

For Decision-Makers: This table answers "What cases need attention right now?" Filter to Overdue = TRUE to see cases past their deadline.

For Analysts: The `Overdue` flag is calculated when the pipeline runs (`today()`). For real-time accuracy, either refresh frequently or calculate overdue status in Power BI using DAX.

8.4.2.2 FactIncident

Grain: One row per incident record.

What It Contains: Core incident information linking students, cases, locations, and dates. This is the primary table for "how many incidents" questions.

```
  FactIncident <- df |>
    select(
      FILE_ID,
      CASE_NUMBER,
      SID,
```

```
    TYPE,
    STATUS,
    ROLE,
    ASSIGNED_TO,
    College,
    INCIDENT_LOCATION,
    INCIDENT_DATE,
    REPORTED_DATE,
    CASE_CREATED_DATE
  )

df_schema(FactIncident) |>
  select(column, type, example) |>
  mutate(across(3, ~ replace(.x, 7, "Hidden"))) |>
  mutate(across(3, ~ replace(.x, 9, "Hidden"))) |>
  print(n = Inf)
```

```
# A tibble: 12 x 3
   column            type      example
   <chr>             <chr>     <chr>
 1 FILE_ID           character 42ee9d11f4bf4611516f7736b5ec4aca
 2 CASE_NUMBER       character c21769dbf6688f856c30587d2e10fd71
 3 SID               character 08e5a2390860f4b7c38a655bf640f122
 4 TYPE              character Non Academic Misconduct
 5 STATUS            character Closed
 6 ROLE              character Respondent
 7 ASSIGNED_TO       character Hidden
 8 College           character DAAP
 9 INCIDENT_LOCATION character Hidden
10 INCIDENT_DATE     Date      2019-09-19
11 REPORTED_DATE     Date      2018-09-20
12 CASE_CREATED_DATE Date      2018-09-25
```

The Baseline Measure

For Decision-Makers: Incidents are typically the most stable unit for trend analysis. Unlike violations (which depend on charging patterns) or sanctions (which depend on outcome patterns), incidents represent "something happened that was reported."

For Analysts: This table connects to most dimensions: DimDate (via INCIDENT_DATE), DimHousing (via INCIDENT_LOCATION), DimStudent (via SID), DimCase (via FILE_ID), and DimHearingOfficer (via ASSIGNED_TO).

8.4.2.3 FactTimeline

Grain: One row per case with timeline metrics.

What It Contains: Process timing measures (adjusted for pause periods). Supports workflow and efficiency analysis.

```
FactTimeline <- df |>
  select(
    FILE_ID,
    INCIDENT_DATE,
    reported_to_case,
    case_to_resolution,
    reported_to_resolution
  )

df_schema(FactTimeline) |>
  select(column, type, example) |>
  print(n = Inf)
```

```
# A tibble: 5 x 3
  column                 type      example
  <chr>                  <chr>     <chr>
1 FILE_ID                character 42ee9d11f4bf4611516f7736b5ec4aca
2 INCIDENT_DATE          Date      2019-09-19
3 reported_to_case       numeric   5
4 case_to_resolution     numeric   9
5 reported_to_resolution numeric   14
```

Understanding Process Flow

For Decision-Makers: Use this to identify bottlenecks. If `reported_to_case` is increasing, there may be intake/triage issues. If `case_to_resolution` is increasing, there

may be hearing capacity issues.

For Analysts: These metrics have already been adjusted for OMIT_RANGES and floored to zero. Join to FactIncident or DimCase on FILE_ID to add case type, assignment, and other context.

8.4.2.4 FactViolation

Grain: One row per alleged violation (charge).

What It Contains: Violations and their outcomes. Supports charging pattern and outcome analysis.

Transformation: Maxient exports violations in wide format (CHARGE_1, CHARGE_2, etc.). This code reshapes them into long format where each row is one violation and outcome.

```
# ──────────────────────────────────────────────────────────────────────────────
# Reshape charges from wide to long format
# ──────────────────────────────────────────────────────────────────────────────
charges <- df |>
  select(FILE_ID, INCIDENT_DATE, College, INCIDENT_LOCATION,
         starts_with("CHARGE_")) |>
  pivot_longer(
    cols = starts_with("CHARGE_"),
    names_to = "ChargeCol",
    values_to = "VIOLATION",
    values_drop_na = FALSE
  ) |>
  mutate(
    ChargeIndex = parse_number(ChargeCol)
  )

# ──────────────────────────────────────────────────────────────────────────────
# Reshape findings to match charges
# ──────────────────────────────────────────────────────────────────────────────
findings <- df |>
  select(FILE_ID, starts_with("FINDING_")) |>
  pivot_longer(
```

```
    cols = starts_with("FINDING_"),
    names_to = "FindingCol",
    values_to = "OUTCOME",
    values_drop_na = FALSE
  ) |>
  mutate(
    ChargeIndex = parse_number(FindingCol)
  ) |>
  select(FILE_ID, ChargeIndex, OUTCOME)

# ──────────────────────────────────────────────────────────────────────────────
# Join charges with findings and add case type
# ──────────────────────────────────────────────────────────────────────────────
FactViolation <- charges |>
  left_join(findings, by = c("FILE_ID", "ChargeIndex")) |>
  left_join(df |> select(FILE_ID, TYPE), by = "FILE_ID") |>

  # Keep only actual violations (not empty rows or excluded types)
  filter(
    !is.na(VIOLATION),
    VIOLATION %nin% c("General Concern", "Aiding and Abetting (Academic)")
  ) |>
  select(
    FILE_ID,
    TYPE,
    INCIDENT_DATE,
    INCIDENT_LOCATION,
    College,
    VIOLATION,
    OUTCOME
  )

# ──────────────────────────────────────────────────────────────────────────────
# Standardize violation names
# ──────────────────────────────────────────────────────────────────────────────
FactViolation <- FactViolation |>
  mutate(VIOLATION = recode(VIOLATION, !!!VIOLATION_REPLACEMENTS,
                            .default = VIOLATION)) |>
  mutate(OUTCOME = recode(OUTCOME,
```

```
                          "Amnesty - No Hearing" = "Amnesty"))

  df_schema(FactViolation) |>
    select(column, type, example) |>
    mutate(across(3, ~ replace(.x, 4, "Hidden"))) |>
    print(n = Inf)
```

```
# A tibble: 7 x 3
  column            type      example
  <chr>             <chr>     <chr>
1 FILE_ID           character 42ee9d11f4bf4611516f7736b5ec4aca
2 TYPE              character Non Academic Misconduct
3 INCIDENT_DATE     Date      2019-09-19
4 INCIDENT_LOCATION character Hidden
5 College           character DAAP
6 VIOLATION         character Alcohol
7 OUTCOME           character Not Responsible
```

Charges vs. Incidents

For Decision-Makers: A single incident might result in three charges (violations). Counting rows in FactViolation tells you how many charges were filed; counting distinct FILE_IDs tells you how many incidents had charges. Both numbers are valid—they answer different questions.

For Analysts: The `ChargeIndex` links each CHARGE_N to its corresponding FINDING_N. The `VIOLATION_REPLACEMENTS` mapping in config.R standardizes violation names (e.g., HTML-encoded ampersands, inconsistent naming). If OUTCOME is NULL, the charge is still pending resolution.

8.4.2.5 FactSanction

Grain: One row per sanction applied.

What It Contains: Sanctions linked to cases. Supports intervention pattern analysis.

Transformation: Similar to violations, sanctions are stored in wide format and reshaped to long format.

```r
FactSanction <- df |>
  select(FILE_ID, TYPE, INCIDENT_DATE, College, ADDENDUM:SUSPENSION) |>

  # Convert "No" values to NA (only "Yes" values should create rows)
  mutate(across(where(is.character), ~ if_else(.x == "No", NA_character_, .x))) |>

  # Reshape from wide to long
  pivot_longer(
    cols = ADDENDUM:SUSPENSION,
    names_to = "Sanction",
    values_to = "Value",
    values_drop_na = TRUE
  ) |>
  select(FILE_ID, TYPE, INCIDENT_DATE, College, Sanction)

df_schema(FactSanction) |>
  select(column, type, example) |>
  print(n = Inf)
```

```
# A tibble: 5 x 3
  column        type      example
  <chr>         <chr>     <chr>
1 FILE_ID       character 42ee9d11f4bf4611516f7736b5ec4aca
2 TYPE          character Non Academic Misconduct
3 INCIDENT_DATE Date      2019-09-19
4 College       character DAAP
5 Sanction      character COMMUNITY_SERVICE
```

Intervention Mix

For Decision-Makers: This table shows what actions we're taking. Join to DimSanction to categorize sanctions (educational, restrictive, etc.) and understand your intervention philosophy in practice.

For Analysts: A case can have multiple sanctions—each becomes a separate row. To count "cases with any educational sanction," you'll need: `FactSanction |> inner_join(DimSanction) |> filter(CATEGORY_LABEL == "Educational") |> distinct(FILE_ID) |> count()`

8.4.2.6 FactRecidivism

Grain: One row per academic year with repeat involvement summary statistics.

What It Contains: Counts of students found responsible and students found responsible in more than one case within the same academic year.

Definition:

Within-Year Recidivism: A student is counted as a recidivist if they were found responsible in **two or more separate cases** (FILE_IDs) within the **same academic year**.

- A student with one case containing multiple responsible findings is **not** a recidivist
- A student with two cases, each with at least one responsible finding, **is** a recidivist
- Recidivism is calculated independently for each academic year (a student can be a recidivist in AY2324 but not in AY2425)

Example:

Student	Cases in AY2324	Responsible Findings	Recidivist?
A	1 case	3 findings (all in same case)	No
B	2 cases	1 finding in each case	Yes
C	3 cases	Findings in 2 of 3 cases	Yes
D	1 case	1 finding	No

```
FactRecidivism <- recidivism(
  df,
  academic_years = ACADEMIC_YEARS
) |>
  select(-Rate)

df_schema(FactRecidivism) |>
  select(column, type, example) |>
  print(n = Inf)
```

```
# A tibble: 3 x 3
  column           type      example
```

```
  <chr>            <chr>     <chr>
1 Academic_Year    character AY1920
2 Found_Resp       integer   475
3 Found_Resp_Again integer   36
```

Using Within-Year Recidivism

For Decision-Makers: This metric answers: "What percentage of students who get in trouble come back to us again within the same academic year?" A rate of 15% means that 15 out of every 100 students found responsible will have another responsible finding before the year ends. This is useful for understanding short-term repeat involvement and the immediate effectiveness of interventions.

For Analysts: The `recidivism()` function counts distinct FILE_IDs with responsible findings per student per academic year. The Rate column is removed here because rates are better calculated in Power BI (allows for dynamic filtering). This measure resets each academic year—a student who was a recidivist in AY2324 starts fresh in AY2425.

8.4.2.7 FactCohortRecidivism

Grain: One row per student whose first responsible finding occurred within the analysis period.

What It Contains: Cohort-based recidivism tracking showing whether each student ever had a subsequent responsible finding after their first.

Definition:

Cohort-Based Recidivism: A student is assigned to a **cohort** based on the academic year of their **first-ever responsible finding**. They are counted as a recidivist if they have **any subsequent case** with a responsible finding, regardless of how much later it occurs.

- Cohort assignment is based on first responsible finding in the dataset
- "Subsequent" means any case with an incident date after the first case's incident date
- There is no time limit—a student who is found responsible for another violation 3 years later still counts
- Each student appears in exactly one cohort (their first year)

Example:

Student	First Responsible Finding	Subsequent Findings	Recidivist?	Cohort
A	AY2122	None	No	AY2122
B	AY2122	AY2324 (2 years later)	Yes	AY2122
C	AY2223	AY2223 (same year) and AY2324	Yes	AY2223
D	AY2425	None yet	No*	AY2425

*Student D may have a violation in the future—their cohort has limited follow-up time.

How This Differs from FactRecidivism:

Aspect	FactRecidivism	FactCohortRecidivism
Question answered	"How many repeat in the same year?"	"How many ever come back?"
Time window	Single academic year	All time after first finding
Grain	One row per academic year	One row per student
A student can appear	In multiple years	In exactly one cohort
Comparable to	Annual repeat rate	Criminal justice recidivism

```
# ──────────────────────────────────────────────────────────────────────────────
# Step 1: Identify all cases with at least one responsible finding
# ──────────────────────────────────────────────────────────────────────────────
finding_vars <- grep("^FINDING_", names(df), value = TRUE)

responsible_cases <- df |>
  filter(ROLE == "Respondent") |>
  mutate(
    has_responsible = if_any(
      all_of(finding_vars),
      ~ tolower(trimws(as.character(.x))) == "responsible"
```

```
    )
  ) |>
  filter(has_responsible) |>
  # One row per case per student
  group_by(SID, FILE_ID, ACADEMIC_YEAR) |>
  summarise(
    INCIDENT_DATE = min(INCIDENT_DATE, na.rm = TRUE),
    .groups = "drop"
  ) |>
  arrange(SID, INCIDENT_DATE)

# ──────────────────────────────────────────────────────────────────────────────
# Step 2: For each student, identify case sequence
# ──────────────────────────────────────────────────────────────────────────────
student_cases <- responsible_cases |>
  group_by(SID) |>
  mutate(
    case_sequence = row_number(),
    total_cases = n()
  ) |>
  ungroup()

# ──────────────────────────────────────────────────────────────────────────────
# Step 3: Extract first case (defines cohort)
# ──────────────────────────────────────────────────────────────────────────────
first_cases <- student_cases |>
  filter(case_sequence == 1) |>
  select(
    SID,
    Cohort_Year = ACADEMIC_YEAR,
    First_Incident_Date = INCIDENT_DATE,
    First_File_ID = FILE_ID,
    Total_Responsible_Cases = total_cases
  )

# ──────────────────────────────────────────────────────────────────────────────
# Step 4: Extract second case (if exists) for recidivism metrics
# ──────────────────────────────────────────────────────────────────────────────
second_cases <- student_cases |>
```

```
  filter(case_sequence == 2) |>
  select(
    SID,
    Second_Incident_Date = INCIDENT_DATE,
    Second_File_ID = FILE_ID,
    Second_Case_Year = ACADEMIC_YEAR
  )

# ──────────────────────────────────────────────────────────────────────────────
# Step 5: Build final fact table
# ──────────────────────────────────────────────────────────────────────────────
FactCohortRecidivism <- first_cases |>
  left_join(second_cases, by = "SID") |>
  mutate(
    Is_Recidivist = !is.na(Second_Incident_Date),
    Days_to_Recidivism = as.numeric(
      difftime(Second_Incident_Date, First_Incident_Date, units = "days")
    ),

    # ──────────────────────────────────────────────────────────────────────────
    # Recidivism intensity categories
    # ──────────────────────────────────────────────────────────────────────────
    Recidivism_Category = case_when(
      Total_Responsible_Cases == 1 ~ "No Recidivism",
      Total_Responsible_Cases == 2 ~ "One Repeat",
      Total_Responsible_Cases == 3 ~ "Two Repeats",
      Total_Responsible_Cases >= 4 ~ "Three+ Repeats"
    ),
    # Factor for proper sorting in visuals
    Recidivism_Category = factor(
      Recidivism_Category,
      levels = c("No Recidivism", "One Repeat", "Two Repeats", "Three+ Repeats")
    ),
    # Numeric version for calculations
    Repeat_Count = Total_Responsible_Cases - 1
  ) |>
  # Add cohort year ordering for proper sorting
  mutate(
    Cohort_AY_Order = as.integer(str_remove(Cohort_Year, "^AY"))
```

```
    ) |>
    # Filter to cohorts within our analysis period
    filter(Cohort_Year %in% ACADEMIC_YEARS) |>
    select(
      SID,
      Cohort_Year,
      Cohort_AY_Order,
      First_Incident_Date,
      Total_Responsible_Cases,
      Repeat_Count,
      Is_Recidivist,
      Recidivism_Category,
      Second_Incident_Date,
      Second_Case_Year,
      Days_to_Recidivism
    ) |>
    arrange(Cohort_Year, First_Incident_Date)

  df_schema(FactCohortRecidivism) |>
    select(column, type, example) |>
    print(n = Inf)
```

```
# A tibble: 11 x 3
   column                  type      example
   <chr>                   <chr>     <chr>
 1 SID                     character bc708c3721e667f4731fdcaa4787db04
 2 Cohort_Year             character AY1920
 3 Cohort_AY_Order         integer   1920
 4 First_Incident_Date     Date      2019-08-15
 5 Total_Responsible_Cases integer   1
 6 Repeat_Count            numeric   0
 7 Is_Recidivist           logical   FALSE
 8 Recidivism_Category     factor    No Recidivism
 9 Second_Incident_Date    Date      2019-09-18
10 Second_Case_Year        character AY1920
11 Days_to_Recidivism      numeric   26
```

Understanding Cohort Recidivism

For Decision-Makers: This table answers: Of the students who first got in trouble in a given year, how many ever got in trouble again? This is how recidivism is reported in criminal justice and other fields, making it useful for benchmarking and for evaluating whether interventions have lasting impact.

Important caveat: Recent cohorts (e.g., AY2425) will show lower recidivism rates simply because less time has passed—not because interventions improved. Always compare cohorts with similar follow-up periods, or note the follow-up time when reporting.

For Analysts: Key measures you can calculate in Power BI: Recidivism Rate and Cohort Size.

Join to DimStudent on SID to disaggregate by demographics. Join to DimDate on First_Incident_Date for time-based filtering.

Key Fields:

Field	Description
SID	Student identifier (anonymized)
Cohort_Year	Academic year of student's first responsible finding
First_Incident_Date	Date of first responsible finding
Total_Responsible_Cases	Total number of cases with responsible findings
Repeat_Count	Number of cases after the first (0 if no recidivism)
Is_Recidivist	TRUE if student had any subsequent responsible finding
Recidivism_Category	Intensity grouping: No Recidivism, One Repeat, Two Repeats, Three+ Repeats
Second_Incident_Date	Date of second responsible finding (NULL if no recidivism)
Second_Case_Year	Academic year of second responsible finding
Days_to_Recidivism	Days between first and second responsible finding (NULL if no recidivism)

Sample Analysis Questions This Enables:

Question	How to Answer
What's our overall recidivism rate?	% where Is_Recidivist = TRUE
Which cohort year has the highest recidivism?	Rate by Cohort_Year (with follow-up time caveat)
How quickly do students recidivate?	Median/distribution of Days_to_Recidivism
Do certain student groups have higher recidivism?	Join to DimStudent, compare rates
Are students with more severe first sanctions less likely to recidivate?	Join first case to FactSanction, compare rates
How intense is repeat involvement?	Distribution of Recidivism_Category
What % of recidivists have 3+ cases?	Filter to Is_Recidivist = TRUE, count Three+ Repeats
What's the average number of repeats among recidivists?	Average Repeat_Count where Is_Recidivist = TRUE

8.4.2.8 FactHousingCensus

Grain: One row per building per term (Fall/Spring only).

What It Contains: Violation counts, census (resident) counts, and calculated violation rates per 100 residents. Supports rate-based housing analysis.

Why Fall/Spring Only: Summer census is typically too small for meaningful rate calculations.

```
# ──────────────────────────────────────────────────────────────────────────────
# Count violations by building and term
# ──────────────────────────────────────────────────────────────────────────────
violation_counts <- FactViolation |>
  left_join(
    DimDate |> select(Date, Academic_Year = `Academic Year`, Semester),
    by = c("INCIDENT_DATE" = "Date")
  ) |>
  filter(Semester %in% c("FA", "SP")) |>
```

```r
  group_by(INCIDENT_LOCATION, Academic_Year, Semester) |>
  summarise(VIOLATIONS = n(), .groups = "drop")

# ──────────────────────────────────────────────────────────────────────────────
# Load and reshape census data
# ──────────────────────────────────────────────────────────────────────────────
census_long <- read_csv(PATHS$imports$housing_census) |>
  rename_with(str_to_upper) |>
  pivot_longer(
    cols = -BUILDING,
    names_to = "Semester",
    values_to = "CENSUS"
  ) |>
  rename(INCIDENT_LOCATION = BUILDING) |>
  mutate(INCIDENT_LOCATION = fix_nbsp(INCIDENT_LOCATION)) |>
  mutate(
    # Map semester codes to academic years
    Academic_Year = case_when(
      Semester %in% c("FA17", "SP18") ~ "AY1718",
      Semester %in% c("FA18", "SP19") ~ "AY1819",
      Semester %in% c("FA19", "SP20") ~ "AY1920",
      Semester %in% c("FA20", "SP21") ~ "AY2021",
      Semester %in% c("FA21", "SP22") ~ "AY2122",
      Semester %in% c("FA22", "SP23") ~ "AY2223",
      Semester %in% c("FA23", "SP24") ~ "AY2324",
      Semester %in% c("FA24", "SP25") ~ "AY2425",
      TRUE ~ "AY2526"
    ),
    # Standardize semester to FA/SP format
    Semester = str_sub(Semester, 1, 2)
  ) |>
  select(INCIDENT_LOCATION, Academic_Year, Semester, CENSUS)

# ──────────────────────────────────────────────────────────────────────────────
# Combine census with violations and calculate rates
# ──────────────────────────────────────────────────────────────────────────────
FactHousingCensus <- census_long |>
  left_join(
    violation_counts,
```

```
      by = c("INCIDENT_LOCATION", "Academic_Year", "Semester")
    ) |>
    left_join(
      DimAcademicTerm,
      by = c("Academic_Year", "Semester")
    ) |>
    mutate(
      # Replace NA with 0 for clean calculations
      CENSUS     = replace_na(CENSUS, 0),
      VIOLATIONS = replace_na(VIOLATIONS, 0),

      # Calculate rate per 100 residents (NULL if no census)
      Violation_Rate = if_else(
        CENSUS > 0,
        round((VIOLATIONS / CENSUS) * 100, 2),
        NA_real_
      )
    ) |>
    select(
      AcademicTermKey,
      INCIDENT_LOCATION,
      Academic_Year,
      Semester,
      CENSUS,
      VIOLATIONS,
      Violation_Rate
    ) |>
    # Exclude temporary housing
    filter(INCIDENT_LOCATION != "MainStay Suites")

  df_schema(FactHousingCensus) |>
    select(column, type, example) |>
    mutate(across(3, ~ replace(.x, 2, "Hidden"))) |>
    print(n = Inf)
```

```
# A tibble: 7 x 3
  column            type      example
  <chr>             <chr>     <chr>
1 AcademicTermKey   numeric   191
2 INCIDENT_LOCATION character Hidden
```

```
3 Academic_Year     character AY1920
4 Semester          character FA
5 CENSUS            numeric   259
6 VIOLATIONS        integer   68
7 Violation_Rate    numeric   26.25
```

Fair Comparisons Across Buildings

For Decision-Makers: A hall with 50 violations and 800 residents (6.25 per 100) is very different from a hall with 50 violations and 200 residents (25 per 100). This table pre-calculates rates so you can compare fairly.

For Analysts: When CENSUS is 0 (building closed or not yet open), Violation_Rate is NULL—not zero. This prevents misleading rate calculations. If a building has violations but no census record, VIOLATIONS will be populated but CENSUS and Violation_Rate will be 0/NULL—investigate the census data.

8.5 Export Star Schema

The final step exports all dimension and fact tables into a single Excel workbook. Power BI imports this workbook, with each worksheet becoming a table.

```
write_xlsx(
  list(
    # ──────────────────────────────────────────────────────────────────────────
    # Dimension Tables
    # ──────────────────────────────────────────────────────────────────────────
    DimStudent        = DimStudent,
    DimCase           = DimCase,
    DimCollege        = DimCollege,
    DimHousing        = DimHousing,
    DimHousingYear    = DimHousingYear,
    DimSanction       = DimSanction,
    DimDate           = DimDate,
    DimAcademicYear   = DimAcademicYear,
    DimHearingOfficer = DimHearingOfficer,
    DimAcademicTerm   = DimAcademicTerm,
```

```
    DimViolation      = DimViolation,

    # ──────────────────────────────────────────────────────────────────────
    # Fact Tables
    # ──────────────────────────────────────────────────────────────────────
    FactIncident         = FactIncident,
    FactTimeline         = FactTimeline,
    FactViolation        = FactViolation,
    FactSanction         = FactSanction,
    FactRecidivism       = FactRecidivism,
    FactCohortRecidivism = FactCohortRecidivism,
    FactDeadline         = FactDeadline,
    FactHousingCensus    = FactHousingCensus
  ),
  PATHS$export
)
```

Single Source of Truth

For Decision-Makers: This workbook is the single source for all conduct dashboards. When it's refreshed, all dashboards update together using consistent definitions.

For Analysts: After import into Power BI, establish relationships:

- Fact tables → DimDate on date fields
- Fact tables → DimCase on FILE_ID
- FactIncident → DimStudent on SID
- FactIncident/FactViolation → DimHousing on INCIDENT_LOCATION
- FactSanction → DimSanction on Sanction
- DimCase → DimHearingOfficer on ASSIGNED_TO

The export path is defined in config.R (`PATHS$export`), making it easy to change output locations without modifying the pipeline code.

9 Computing Environment

This section documents the software environment used to run the pipeline, supporting reproducibility and troubleshooting.

```
- Session info ---------------------------------------------------------------
 setting  value
 version  R version 4.5.2 (2025-10-31 ucrt)
 os       Windows 11 x64 (build 26100)
 system   x86_64, mingw32
 ui       RTerm
 language (EN)
 collate  English_United States.utf8
 ctype    English_United States.utf8
 tz       America<path>
 date     2026-01-20
 pandoc   3.6.3 @ <path> Files<path> (via rmarkdown)
 quarto   1.8.26 @ <path>

- Packages -------------------------------------------------------------------
 package   * version date (UTC) lib source
 digest    * 0.6.39  2025-11-19 [1] CRAN (R 4.5.1)
 dplyr     * 1.1.4   2023-11-17 [1] CRAN (R 4.5.2)
 forcats   * 1.0.1   2025-09-25 [1] CRAN (R 4.5.2)
 ggplot2   * 4.0.1   2025-11-14 [1] CRAN (R 4.5.2)
 lubridate * 1.9.4   2024-12-08 [1] CRAN (R 4.5.2)
 openssl   * 2.3.4   2025-09-30 [1] CRAN (R 4.5.2)
 purrr     * 1.2.1   2026-01-09 [1] CRAN (R 4.5.2)
 readr     * 2.1.6   2025-11-14 [1] CRAN (R 4.5.2)
 stringr   * 1.6.0   2025-11-04 [1] CRAN (R 4.5.2)
 tibble    * 3.3.1   2026-01-11 [1] CRAN (R 4.5.2)
 tidyr     * 1.3.2   2025-12-19 [1] CRAN (R 4.5.2)
 tidyverse * 2.0.0   2023-02-22 [1] CRAN (R 4.5.2)
 writexl   * 1.5.4   2025-04-15 [1] CRAN (R 4.5.2)

 [1] <path>
 [2] <path> Files<path>
 * -- Packages attached to the search path.
```

---

10 About the Author

Josh Moermond serves as the Assistant Director for Residential Conduct and Assessment at the University of Cincinnati. He has spent nearly a decade working at the intersection of student conduct administration and residence life.

Moermond's work emerged from a problem he observed early in his career: student conduct and residence life offices were generating vast amounts of data but lacked the systems to extract meaningful insights. As a residence life professional turned conduct administrator, he recognized that the field wasn't suffering from a lack of information—it was asking the wrong questions and missing the tools to answer the right ones.

This realization led him to develop a comprehensive analytics pipeline that transforms raw Maxient case data into actionable intelligence using R and Power BI. The resulting dashboard system tracks case volume, lifecycle timelines, demographic patterns, violation trends, recidivism rates, and much more, enabling his team to make evidence-based decisions about behavioral trends, resource allocation, and community safety.

Moermond's technical approach is grounded in practitioner needs rather than abstract data science principles. He is not a trained data scientist but a student conduct professional who learned to program out of necessity—a perspective that shapes both his methodology and his belief that assessment excellence is accessible to all conduct administrators, regardless of technical background.

Beyond his analytical work, Moermond maintains expertise in case management, threat assessment, and behavioral intervention. He holds certifications from the National Association for Behavioral Intervention and Threat Assessment (NABITA) in threat management, case management advanced interventions, violence risk assessment of the written word, and assessment and management of suicidal ideation.

Moermond earned his Master of Arts in Educational Leadership from Eastern Michigan University and a Graduate Certificate in Data Analytics at the University of Cincinnati. He has contributed to grant applications through data-driven research on student behavioral trends.

He can be reached at moermondsahe@gmail.com or via LinkedIn.

11 License and Attribution

Recommended citation:

Moermond, J. L. (2026). Building an Assessment-Ready Conduct Data Model: Transforming Maxient Exports into Reliable Dashboards for Student Conduct Analysis.

Intended Use:

This work is intended to support professional practice, assessment, and institutional learning in student conduct and student affairs contexts. Institutions are encouraged to adapt the approach to their own data structures, policies, and governance requirements. Reuse of this work should preserve its assessment-oriented intent and respect applicable ethical, legal, and student privacy standards.

12 Use of Generative AI

Portions of this document were developed with the assistance of **generative AI tools.**

Generative AI was used as a writing and technical support aid, including:

- Refining language for clarity and accessibility
- Improving organization and flow of explanatory text
- Assisting with code errors, readability, formatting, custom functions, and documentation
- Helping draft example explanations, tables, and narrative framing

All analytic decisions, data transformations, methodological choices, and interpretations were designed, reviewed, and validated by the author. The underlying logic of the data model, transformation pipeline, and reporting framework reflects professional judgment grounded in student conduct practice and assessment standards.

No confidential, proprietary, or student-level data was shared with generative AI systems as part of this work.

The author retains full responsibility for the content, accuracy, and conclusions presented.

13 References

Eddelbuettel, Dirk. 2025. *digest: Create Compact Hash Digests of r Objects*. https://doi.org/10.32614/CRAN.package.digest.

Moermond, Joshua L. 2026. "Conduct_funcs: Custom r Functions for Student Conduct Analytics." Unpublished internal R script.

Ooms, Jeroen. 2025a. *Openssl: Toolkit for Encryption, Signatures and Certificates Based on OpenSSL*. https://doi.org/10.32614/CRAN.package.openssl.

———. 2025b. *Writexl: Export Data Frames to Excel 'Xlsx' Format*. https://doi.org/10.32614/CRAN.package.writexl.

R Core Team. 2025. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Wickham, Hadley, Mara Averick, Jennifer Bryan, Winston Chang, Lucy D'Agostino McGowan, Romain François, Garrett Grolemund, et al. 2019. "Welcome to the tidyverse." *Journal of Open Source Software* 4 (43): 1686. https://doi.org/10.21105/joss.01686.

Wickham, Hadley, Winston Chang, Robert Flight, Kirill Müller, and Jim Hester. 2026. *Sessioninfo: R Session Information*. https://github.com/r-lib/sessioninfo.

A Entity-Relationship Diagram

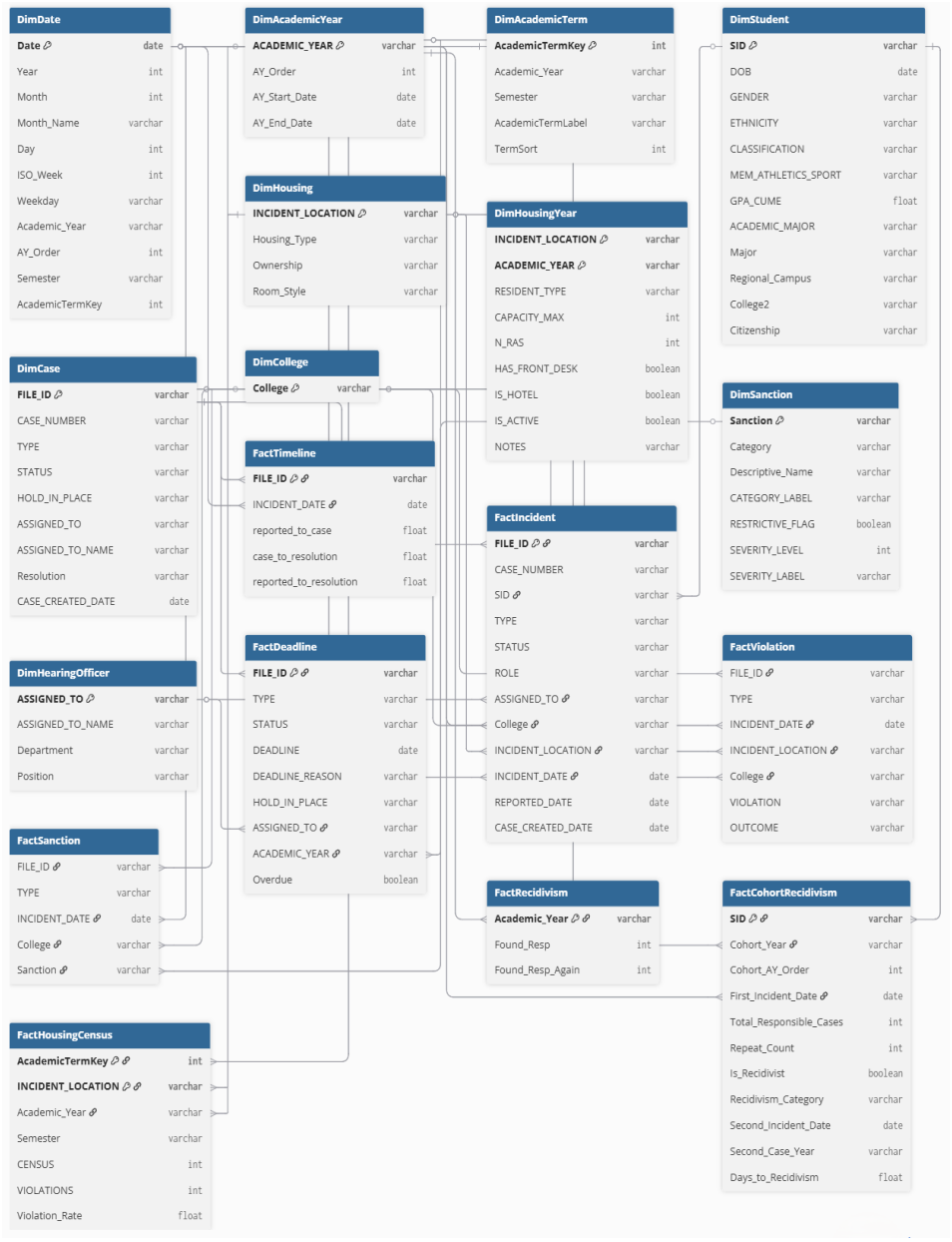

Figure 1: Star Schema Entity-Relationship Diagram

B Custom Functions (conduct_funcs.R)

This file contains reusable functions for data transformation, anonymization, and analysis.

```r
# conduct_funcs.R
# Custom functions for student conduct data pipeline
# Author: Joshua L. Moermond
# Last Updated: 2026-01-13

# Load required packages for function operations
library(dplyr)
library(tidyr)
library(lubridate)
library(stringr)
library(purrr)

# Utility Functions ----------------------------------------------------

#' Display schema of a data frame
#' @param df Data frame to describe
#' @param show_example Logical; whether to show example values
#' @param max_example_length Maximum characters for example values
#' @return Data frame describing the schema
#' @export
df_schema <- function(df, show_example = TRUE, max_example_length = 60) {
  schema <- tibble(
    column = names(df),
    type = map_chr(df, ~ paste(class(.x), collapse = ", ")),
    non_na_count = map_int(df, ~ sum(!is.na(.x))),
    na_count = map_int(df, ~ sum(is.na(.x))),
    na_percent = round((na_count / nrow(df)) * 100, 1)
  )

  if (show_example) {
    schema <- schema |>
      mutate(
        example = map_chr(df, function(col) {
          val <- col[!is.na(col)][1]
          # Handle empty columns OR NA result
          if (length(val) == 0 || is.na(val)) {
            return(NA_character_)
          }
          ex <- as.character(val)
          if (nchar(ex) > max_example_length) {
            paste0(substr(ex, 1, max_example_length - 3), "...")
          } else {
            ex
```

```
          }
        })
      )
  }

  schema
}

#' Validate required columns exist in dataframe
#' @param df Data frame to validate
#' @param required_cols Character vector of required column names
#' @param df_name Name of dataframe for error messages
#' @return Invisibly returns TRUE if valid, stops with error if not
#' @export
validate_columns <- function(df, required_cols, df_name = "dataframe") {
  missing <- setdiff(required_cols, names(df))

  if (length(missing) > 0) {
    stop(
      "Required columns missing from ", df_name, ":\n  ",
      paste(missing, collapse = ", "),
      call. = FALSE
    )
  }

  invisible(TRUE)
}

#' Execute a function with progress reporting
#' @param desc Description of the operation
#' @param expr Expression to evaluate
#' @return Result of the expression
#' @export
with_progress <- function(desc, expr) {
  if (requireNamespace("cli", quietly = TRUE)) {
    cli::cli_progress_step(desc)
    result <- force(expr)
    cli::cli_progress_done()
    result
  } else {
    message(desc, "...")
    force(expr)
  }
}

#' Negate %in% operator
#' @export
`%nin%` <- Negate(`%in%`)
```

```r
# Date and Time Functions ---------------------------------------------

#' Convert columns to Date format
#' @param df Data frame
#' @param columns Character vector of column names
#' @return Data frame with converted date columns
#' @export
convert_to_date <- function(df, columns) {
  # Validate columns exist
  missing_cols <- setdiff(columns, names(df))
  if (length(missing_cols) > 0) {
    warning("Columns not found: ", paste(missing_cols, collapse = ", "))
    columns <- intersect(columns, names(df))
  }

  df |>
    mutate(across(
      all_of(columns),
      ~ {
        if (inherits(.x, "Date")) {
          .x  # Already a date, return as-is
        } else {
          # Try multiple date formats
          as_date(parse_date_time(.x, orders = c("ymd", "mdy", "dmy"),
                                  quiet = TRUE))
        }
      }
    ))
}

#' Calculate academic year from a date
#' @param date Date vector
#' @return Character vector of academic years (e.g., "AY2324")
#' @export
#' @examples
#' compute_academic_year(as.Date("2023-09-15"))  # Returns "AY2324"
#' compute_academic_year(as.Date("2024-05-10"))  # Returns "AY2324"
compute_academic_year <- function(date) {
  year_num <- year(date)
  month_num <- month(date)

  # Academic year starts in August
  # Aug-Dec: current calendar year is AY start
  # Jan-Jul: previous calendar year is AY start
  ay_start_year <- if_else(month_num >= 8, year_num, year_num - 1)

  paste0(
    "AY",
```

```
    substr(ay_start_year, 3, 4),
    substr(ay_start_year + 1, 3, 4)
  )
}

#' Compute overlap days between date ranges and omit periods
#' @param start_dates Vector of start dates
#' @param end_dates Vector of end dates
#' @param resolution Vector of resolution types
#' @param omit_ranges List of date range pairs to exclude
#' @return Numeric vector of overlap days
#' @export
compute_overlaps <- function(start_dates, end_dates, resolution,
                             omit_ranges) {
  # Validate inputs
  n <- length(start_dates)
  if (length(end_dates) != n || length(resolution) != n) {
    stop("start_dates, end_dates, and resolution must have same length")
  }

  # Create data frame
  date_df <- tibble(
    start = ymd(start_dates),
    end = ymd(end_dates),
    resolution = resolution,
    adjustment = 0
  )

  # Only adjust non-"Warning Letter" cases
  adjust_mask <- date_df$resolution != "Warning Letter"

  # Calculate overlaps for each omit range
  for (omit_range in omit_ranges) {
    omit_start <- ymd(omit_range[1])
    omit_end <- ymd(omit_range[2])

    # Calculate overlap only where needed
    overlap_start <- pmax(date_df$start[adjust_mask], omit_start,
                          na.rm = TRUE)
    overlap_end <- pmin(date_df$end[adjust_mask], omit_end,
                        na.rm = TRUE)

    # Days of overlap (inclusive, so +1)
    overlap_days <- pmax(0, as.numeric(overlap_end - overlap_start) + 1)

    # Accumulate adjustments
    date_df$adjustment[adjust_mask] <-
      date_df$adjustment[adjust_mask] + overlap_days
  }
```

```r
  date_df$adjustment
}


# Data Cleaning Functions ---------------------------------------------

#' Replace charges using vectorized operations
#' @param df Data frame
#' @param charge_cols Character vector of column names to process
#' @param replacements Named list of replacements
#' @return Data frame with replaced values
#' @export
replace_charges <- function(df, charge_cols, replacements) {
  # Validate inputs
  missing_cols <- setdiff(charge_cols, names(df))
  if (length(missing_cols) > 0) {
    stop("Columns not found in dataframe: ",
         paste(missing_cols, collapse = ", "))
  }

  # Vectorized replacement using case_match
  df |>
    mutate(across(
      all_of(charge_cols),
      ~ case_match(
        .x,
        !!!replacements,
        .default = .x
      )
    ))
}

#' Replace NA values with empty strings
#' @param df Data frame
#' @param columns Character vector of column names
#' @return Data frame with NA values replaced
#' @export
replace_na_empty <- function(df, columns) {
  df |>
    mutate(across(
      all_of(columns),
      ~ if_else(is.na(.x), "", as.character(.x))
    ))
}


# Anonymization Functions ---------------------------------------------
```

```r
#' Vectorized anonymization function using HMAC
#' @param x Vector to hash
#' @param key Encryption key (raw vector)
#' @param algo Hashing algorithm (default: "sha256")
#' @param n_hex Number of hex characters to return
#' @return Character vector of hashed values
#' @export
hash_col <- function(x, key, algo = "sha256", n_hex = 32) {
  vapply(x, function(val) {
    if (is.na(val)) return(NA_character_)
    val_chr <- as.character(val)
    h <- openssl::sha256(paste0(val_chr, rawToChar(key)), key = key)
    if (!is.null(n_hex)) substr(h, 1, n_hex) else h
  }, character(1), USE.NAMES = FALSE)
}


# Analysis Functions ------------------------------------------------

#' Calculate recidivism rates with optional grouping
#' @param df Data frame containing conduct records
#' @param academic_years Character vector of academic years
#' @param group_by Character vector of grouping variables (e.g., "College")
#' @param format Output format: "display" (with %) or "pbi" (numeric Rate)
#' @return Data frame with recidivism metrics
#' @export
calculate_recidivism <- function(df,
                                 academic_years,
                                 group_by = NULL,
                                 format = c("display", "pbi")) {
  format <- match.arg(format)

  # Get FINDING columns
  finding_vars <- grep("^FINDING_", names(df), value = TRUE)
  if (length(finding_vars) == 0) {
    stop("No FINDING_* columns found in dataframe")
  }

  # Validate required columns
  validate_columns(df, c("ROLE", "SID", "FILE_ID", "INCIDENT_DATE"),
                   "conduct data")

  # If no ACADEMIC_YEAR column, compute it
  if (!"ACADEMIC_YEAR" %in% names(df)) {
    df <- df |> mutate(ACADEMIC_YEAR =
                         compute_academic_year(INCIDENT_DATE))
  }
```

```r
# Standardize grouping columns
if (!is.null(group_by)) {
  df <- df |>
    mutate(across(all_of(group_by),
                  ~ replace_na(as.character(.x), "Not Reported")))
}

# ──────────────────────────────────────────────────────────────────────────
# Step 1: Identify FILE_IDs with at least one responsible finding
# ──────────────────────────────────────────────────────────────────────────
cases_with_responsible <- df |>
  filter(
    ACADEMIC_YEAR %in% academic_years,
    ROLE == "Respondent"
  ) |>
  mutate(
    has_responsible = if_any(
      all_of(finding_vars),
      ~ tolower(trimws(as.character(.x))) == "responsible"
    )
  ) |>
  filter(has_responsible) |>
  # Get one row per case per student, with earliest incident date
  group_by(ACADEMIC_YEAR, SID, FILE_ID) |>
  summarise(
    INCIDENT_DATE = min(INCIDENT_DATE, na.rm = TRUE),
    .groups = "drop"
  )

# ──────────────────────────────────────────────────────────────────────────
# Step 2: Count distinct cases per student per academic year
# ──────────────────────────────────────────────────────────────────────────
cases_per_student <- cases_with_responsible |>
  group_by(ACADEMIC_YEAR, SID) |>
  summarise(
    case_count = n_distinct(FILE_ID),
    .groups = "drop"
  )

# ──────────────────────────────────────────────────────────────────────────
# Step 3: Summarize by academic year (and optional grouping)
# ──────────────────────────────────────────────────────────────────────────

# If grouping, we need to bring group info back in
if (!is.null(group_by)) {
  # Get the grouping value for each student (use most recent)
  student_groups <- df |>
    filter(ACADEMIC_YEAR %in% academic_years, ROLE == "Respondent") |>
```

```
    arrange(SID, desc(INCIDENT_DATE)) |>
    group_by(SID) |>
    slice_head(n = 1) |>
    ungroup() |>
    select(SID, all_of(group_by))

  cases_per_student <- cases_per_student |>
    left_join(student_groups, by = "SID")
}

group_vars <- c("ACADEMIC_YEAR", group_by)

result <- cases_per_student |>
  group_by(across(all_of(group_vars))) |>
  summarise(
    Found_Resp = n(),
    Found_Resp_Again = sum(case_count > 1),
    Rate = if_else(Found_Resp > 0,
                   Found_Resp_Again / Found_Resp,
                   NA_real_),
    .groups = "drop"
  ) |>
  rename(Academic_Year = ACADEMIC_YEAR)

# Add overall row if grouped
if (!is.null(group_by)) {
  overall <- cases_per_student |>
    group_by(ACADEMIC_YEAR) |>
    summarise(
      Found_Resp = n(),
      Found_Resp_Again = sum(case_count > 1),
      Rate = if_else(Found_Resp > 0, Found_Resp_Again / Found_Resp,
                     NA_real_),
      .groups = "drop"
    ) |>
    mutate(!!group_by := "Overall") |>
    rename(Academic_Year = ACADEMIC_YEAR)

  result <- bind_rows(result, overall)
}

# Format output
if (format == "display") {
  result <- result |>
    mutate(Rate_Display = scales::percent(Rate, accuracy = 0.01)) |>
    select(-Rate) |>
    rename(Rate = Rate_Display)
} else if (format == "pbi") {
```

```
    if (!is.null(group_by)) {
      complete_grid <- expand_grid(
        Academic_Year = academic_years,
        !!sym(group_by) := unique(result[[group_by]])
      )
      result <- complete_grid |>
        left_join(result, by = c("Academic_Year", group_by))
    }

    result <- result |>
      mutate(
        AY_Order = as.integer(str_remove(Academic_Year, "^AY")),
        Rate_Pct_Label = if_else(is.na(Rate), NA_character_,
                                 scales::percent(Rate, accuracy = 0.01))
      )

    if (!is.null(group_by)) {
      result <- result |>
        arrange(AY_Order, desc(!!sym(group_by) == "Overall"),
                !!sym(group_by))
    } else {
      result <- result |> arrange(AY_Order)
    }
  }

  result
}

#' Calculate cohort-based recidivism rates
#'
#' Defines cohorts by the academic year of each student's
#' first responsible finding,
#' then calculates what percentage had any subsequent responsible finding.
#'
#' @param df Data frame containing conduct records
#' @param cohort_years Character vector of academic years to use as cohorts
#' @param followup_through Academic year to track recidivism through
#' (default: latest in data)
#' @param group_by Optional grouping variable (e.g., "College")
#' @param format Output format: "display" (with %) or "pbi" (numeric Rate)
#' @return Data frame with cohort-based recidivism metrics
#' @export
#' @examples
#' # What % of students whose first responsible finding was in AY2122
#' # had another responsible finding in AY2223, AY2324, or AY2425?
#' calculate_cohort_recidivism(df, cohort_years = c("AY2122", "AY2223",
#' "AY2324"))
calculate_cohort_recidivism <- function(df,
```

```
                                    cohort_years,
                                    followup_through = NULL,
                                    group_by = NULL,
                                    format = c("display", "pbi")) {
format <- match.arg(format)

# Get FINDING columns
finding_vars <- grep("^FINDING_", names(df), value = TRUE)
if (length(finding_vars) == 0) {
  stop("No FINDING_* columns found in dataframe")
}

# Validate required columns
validate_columns(df, c("ROLE", "SID", "FILE_ID", "INCIDENT_DATE",
                       "ACADEMIC_YEAR"),
                 "conduct data")

# Standardize grouping columns
if (!is.null(group_by)) {
  df <- df |>
    mutate(across(all_of(group_by),
                  ~ replace_na(as.character(.x), "Not Reported")))
}

# ──────────────────────────────────────────────────────────────────────
# Step 1: Identify all cases with at least one responsible finding
# ──────────────────────────────────────────────────────────────────────
responsible_cases <- df |>
  filter(ROLE == "Respondent") |>
  mutate(
    has_responsible = if_any(
      all_of(finding_vars),
      ~ tolower(trimws(as.character(.x))) == "responsible"
    )
  ) |>
  filter(has_responsible) |>
  group_by(SID, FILE_ID, ACADEMIC_YEAR) |>
  summarise(
    INCIDENT_DATE = min(INCIDENT_DATE, na.rm = TRUE),
    .groups = "drop"
  )

# ──────────────────────────────────────────────────────────────────────
# Step 2: For each student, identify their FIRST responsible case ever
# ──────────────────────────────────────────────────────────────────────
first_responsible <- responsible_cases |>
  arrange(SID, INCIDENT_DATE) |>
  group_by(SID) |>
```

```
  slice_head(n = 1) |>
  ungroup() |>
  rename(
    first_file_id = FILE_ID,
    first_incident_date = INCIDENT_DATE,
    cohort_year = ACADEMIC_YEAR
  )

# ──────────────────────────────────────────────────────────────────
# Step 3: Identify students who had ANY subsequent responsible case
# ──────────────────────────────────────────────────────────────────
subsequent_cases <- responsible_cases |>
  inner_join(
    first_responsible |> select(SID, first_incident_date),
    by = "SID"
  ) |>
  filter(INCIDENT_DATE > first_incident_date)

# Apply followup cutoff if specified
if (!is.null(followup_through)) {
  # Get end date of followup year
  followup_end <- as.Date(paste0("20", substr(followup_through, 5, 6),
                                 "-07-31"))
  subsequent_cases <- subsequent_cases |>
    filter(INCIDENT_DATE <= followup_end)
}

students_with_recidivism <- subsequent_cases |>
  distinct(SID) |>
  mutate(is_recidivist = TRUE)

# ──────────────────────────────────────────────────────────────────
# Step 4: Build cohort summary
# ──────────────────────────────────────────────────────────────────

# Filter to requested cohort years
cohort_data <- first_responsible |>
  filter(cohort_year %in% cohort_years) |>
  left_join(students_with_recidivism, by = "SID") |>
  mutate(is_recidivist = replace_na(is_recidivist, FALSE))

# Add grouping variable if specified
if (!is.null(group_by)) {
  student_groups <- df |>
    filter(ROLE == "Respondent") |>
    arrange(SID, desc(INCIDENT_DATE)) |>
    group_by(SID) |>
    slice_head(n = 1) |>
```

```
    ungroup() |>
    select(SID, all_of(group_by))

  cohort_data <- cohort_data |>
    left_join(student_groups, by = "SID")
}

# ──────────────────────────────────────────────────────────────────────
# Step 5: Summarize by cohort year (and optional grouping)
# ──────────────────────────────────────────────────────────────────────
group_vars <- c("cohort_year", group_by)

result <- cohort_data |>
  group_by(across(all_of(group_vars))) |>
  summarise(
    Cohort_N = n(),
    Recidivists = sum(is_recidivist),
    Rate = if_else(Cohort_N > 0,
                   Recidivists / Cohort_N,
                   NA_real_),
    .groups = "drop"
  ) |>
  rename(Cohort_Year = cohort_year)

# Add overall row if grouped
if (!is.null(group_by)) {
  overall <- cohort_data |>
    group_by(cohort_year) |>
    summarise(
      Cohort_N = n(),
      Recidivists = sum(is_recidivist),
      Rate = if_else(Cohort_N > 0, Recidivists / Cohort_N, NA_real_),
      .groups = "drop"
    ) |>
    mutate(!!group_by := "Overall") |>
    rename(Cohort_Year = cohort_year)

  result <- bind_rows(result, overall)
}

# ──────────────────────────────────────────────────────────────────────
# Step 6: Format output
# ──────────────────────────────────────────────────────────────────────
if (format == "display") {
  result <- result |>
    mutate(Rate_Display = scales::percent(Rate, accuracy = 0.1)) |>
    select(-Rate) |>
    rename(Rate = Rate_Display)
} else if (format == "pbi") {
```

```
    if (!is.null(group_by)) {
      complete_grid <- expand_grid(
        Cohort_Year = cohort_years,
        !!sym(group_by) := unique(result[[group_by]])
      )
      result <- complete_grid |>
        left_join(result, by = c("Cohort_Year", group_by))
    }

    result <- result |>
      mutate(
        AY_Order = as.integer(str_remove(Cohort_Year, "^AY")),
        Rate_Pct_Label = if_else(is.na(Rate), NA_character_,
                                 scales::percent(Rate, accuracy = 0.1))
      )

    if (!is.null(group_by)) {
      result <- result |>
        arrange(AY_Order, desc(!!sym(group_by) == "Overall"),
                !!sym(group_by))
    } else {
      result <- result |> arrange(AY_Order)
    }
  }

  result
}

#' Calculate year-over-year violation comparison
#' @param df Data frame containing conduct records
#' @param years Character vector of academic years to compare
#' @return Data frame with violation counts and percent changes
#' @export
charge_comp <- function(df, years) {
  # Initialize list to store yearly data
  yearly_data <- list()

  # Process each year
  for (year in years) {
    year_col_name <- paste0("AY", substr(year, 3, 6))

    yearly_data[[year]] <- df |>
      filter(ACADEMIC_YEAR == year) |>
      select(CHARGE_1:CHARGE_6) |>
      pivot_longer(
        cols = CHARGE_1:CHARGE_6,
        names_to = 'CHARGE',
        values_to = 'VIOLATION',
```

```
        values_drop_na = TRUE
      ) |>
      count(VIOLATION) |>
      mutate(n = replace_na(n, 0)) |>
      rename(!!year_col_name := n)
  }

  # Merge all yearly data
  y2y_comp <- reduce(yearly_data, full_join, by = "VIOLATION")

  # Replace NA with 0 and calculate percentage changes
  for (i in seq_along(years)) {
    year_col_name <- paste0("AY", substr(years[i], 3, 6))
    y2y_comp[[year_col_name]] <- replace_na(y2y_comp[[year_col_name]], 0)

    if (i > 1) {
      prev_year_col_name <- paste0("AY", substr(years[i - 1], 3, 6))
      change_col_name <- paste0("Change from ", prev_year_col_name,
                                " to ", year_col_name)
      y2y_comp[[change_col_name]] <- scales::percent(
        (y2y_comp[[year_col_name]] - y2y_comp[[prev_year_col_name]]) /
          y2y_comp[[prev_year_col_name]],
        accuracy = 0.01
      )
    }
  }

  y2y_comp |>
    filter(!is.na(VIOLATION)) |>
    arrange(VIOLATION)
}


# Legacy function wrappers for backward compatibility -----------------------

#' Calculate recidivism for given academic years (legacy wrapper)
#' @param df Data frame containing conduct records
#' @param academic_years Character vector of academic years
#' @return Data frame with recidivism rates
#' @export
recidivism <- function(df, academic_years) {
  calculate_recidivism(df, academic_years, format = "display")
}

#' Calculate recidivism by college (legacy wrapper)
#' @param df Data frame containing conduct records
#' @param academic_years Character vector of academic years
#' @return Data frame with recidivism rates by college
#' @export
```

```r
recidivism_by_college <- function(df, academic_years) {
  calculate_recidivism(df, academic_years, group_by = "College",
                       format = "display")
}

#' Calculate recidivism by college for Power BI (legacy wrapper)
#' @param df Data frame containing conduct records
#' @param academic_years Character vector of academic years
#' @return Data frame with recidivism rates by college formatted
#' for Power BI
#' @export
recidivism_by_college_pbi <- function(df, academic_years) {
  calculate_recidivism(df, academic_years, group_by = "College",
                       format = "pbi")
}

fix_nbsp <- function(x) {
  x <- as.character(x)
  x <- gsub("\u00A0", " ", x, fixed = TRUE)  # NBSP -> normal space
  x <- gsub("[[:space:]]+", " ", x)          # collapse repeated whitespace
  trimws(x)
}
```

C Configuration File (config.R)

This file contains all constants, mappings, and settings used by the pipeline. Update this file to add new academic years, modify location mappings, or adjust pause periods.

```
# config.R
# Configuration file for student conduct data pipeline
# Contains all constants, mappings, and settings
# Author: Joshua L. Moermond
# Last Updated: 2026-01-13

# Academic Years ------------------------------------------------------------

ACADEMIC_YEARS <- c(
  "AY1920",
  "AY2021",
  "AY2122",
  "AY2223",
  "AY2324",
  "AY2425",
  "AY2526"
)


# Date Ranges ---------------------------------------------------------------

DATE_RANGE <- list(
  start = as.Date("2019-08-01"),
  end = as.Date("2027-07-31")
```

```
)


# File Paths ------------------------------------------------------------

PATHS <- list(
  export = "Exports/REDConduct_StarSchema.xlsx",
  pepper = "Imports/pepper.bin",
  imports = list(
    export1 = "Imports/maxientExport_082019_072021.csv",
    export2 = "Imports/maxientExport_082021_072024.csv",
    export3 = "Imports/maxientExport_082024_072027.csv",
    academic_plans = "Imports/academic_plans.csv",
    hearing_officers = "Imports/hearing_officers.csv",
    dim_housing = "Imports/DimHousing.csv",
    dim_housing_year = "Imports/DimHousingYear.csv",
    dim_sanction = "Imports/DimSanction.csv",
    housing_census = "Imports/housing_census.csv",
    dim_charge = "Imports/DimCharge.csv"
  )
)


# Residential Locations (RED) -------------------------------------------

RED_LOCATIONS <- c(
  "101 East Corry",
  "Bellevue Gardens",
  "Calhoun Hall",
  "Comfort Inn",
```

```
  "CP Cincy",
  "CRC Hall",
  "Dabney Hall",
  "Daniels Hall",
  "Fairfield Inn",
  "Gateway Lofts",
  "Hampton Inn",
  "Jefferson House",
  "Marian Spencer Hall",
  "Morgens Hall",
  "Schneider Hall",
  "Scioto Hall",
  "Senator Place",
  "Siddall Hall",
  "Stetson Square",
  "Stratford Heights",
  "The Deacon",
  "The Eden",
  "The Graduate",
  "The Union",
  "Turner Hall",
  "University Edge",
  "University Park Apartments",
  "USquare",
  "The Verge"
)


# Location Name Mappings ------------------------------------------------
```

```
LOCATION_MAP <- c(
  "101 Corry" = "101 East Corry",
  "Campus Park Apartments" = "CP Cincy",
  "Campus Rec Cen Residence Hall" = "CRC Hall",
  "Cp Cincy" = "CP Cincy",
  "Crc Hall" = "CRC Hall",
  "Deacon" = "The Deacon",
  "Fairfield Inn &Amp; Suites" = "Fairfield Inn",
  "Graduate Hotel" = "The Graduate",
  "Hampton Inn &Amp; Suites Cincinnati/Uptown University" = "Hampton Inn",
  "Hampton Inn &Amp; Suites" = "Hampton Inn",
  "Stratford Heights Bld 1" = "Stratford Heights",
  "Stratford Heights Bld 4" = "Stratford Heights",
  "Stratford Heights Bldg 5" = "Stratford Heights",
  "Stratford Heights Bldg 9" = "Stratford Heights",
  "Stratford Hts Bld 10" = "Stratford Heights",
  "Stratford Hts Bld 12 Tower Hall" = "Stratford Heights",
  "Stratford Hts Bld 2" = "Stratford Heights",
  "Stratford Hts Bld 3" = "Stratford Heights",
  "The Deacon Apartments" = "The Deacon",
  "The Eden Apartments" = "The Eden",
  "The Graduate Cincinnati" = "The Graduate",
  "University Park Apts" = "University Park Apartments",
  "University Park Apts South" = "University Park Apartments",
  "University Park Apts. (Calhoun)" = "University Park Apartments",
  "Usquare" = "USquare",
  "Verge" = "The Verge",
  "The Comfort Inn" = "Comfort Inn",
  "MainStay Suites" = "Comfort Inn"
)
```

```
# Violation Name Standardization ----------------------------------------

VIOLATION_REPLACEMENTS <- c(
  "Academic Misconduct - Aiding &amp;amp; Abetting" =
    "Academic Misconduct - Aiding and Abetting",
  "Academic Misconduct - Violating Ethical or Professional Standards" =
    "Academic Misconduct - Violating Standards",
  "Aiding &amp;amp; Abetting" =
    "Aiding and Abetting",
  "Alcohol - Underage possession" =
    "Alcohol",
  "Alcohol - Public Intoxication" =
    "Alcohol",
  "Dishonesty &amp;amp; Misrepresentation" =
    "Dishonesty and Misrepresentation",
  "Drugs or Narcotics - Paraphernalia" =
    "Drugs or Narcotics",
  "Drugs or Narcotics - Possession/ Use" =
    "Drugs or Narcotics",
  "Drugs or Narcotics - Distribution" =
    "Drugs or Narcotics",
  "Drugs or Narcotics - Unauthorized prescription" =
    "Drugs or Narcotics",
  "Physical Abuse or Harm, or Threat of Physical Abuse or Harm" =
    "Physical Abuse or Harm (or Threat)",
  "Residence Hall Rules &amp;amp; Regulations - Appliances and Electric Cords" =
    "GUL - Appliances and Electric Cords",
  "Residence Hall Rules &amp;amp; Regulations - Commercial &amp;amp; Business Activity" =
    "GUL - Commercial and Business Activity",
```

```
"Residence Hall Rules &amp;amp; Regulations - Dining Centers" =
  "GUL - Dining Centers",
"Residence Hall Rules &amp;amp; Regulations - Elevators, Hallways &amp;amp; Restricted Areas" =
  "GUL - Elevators, Hallways, and Restricted Areas",
"Residence Hall Rules &amp;amp; Regulations - Fire Safety" =
  "GUL - Fire Safety",
"Residence Hall Rules &amp;amp; Regulations - Furniture" =
  "GUL - Furniture",
"Residence Hall Rules &amp;amp; Regulations - Gambling" =
  "GUL - Gambling",
"Residence Hall Rules &amp;amp; Regulations - Guests" =
  "GUL - Guests",
"Residence Hall Rules &amp;amp; Regulations - Health &amp;amp; Safety" =
  "GUL - Health and Safety",
"Residence Hall Rules &amp;amp; Regulations - Keys &amp;amp; Access" =
  "GUL - Keys and Access",
"Residence Hall Rules &amp;amp; Regulations - Mail" =
  "GUL - Mail",
"Residence Hall Rules &amp;amp; Regulations - Pets, Service Animals, &amp;amp; Assistance Animals" =
  "GUL - Pets, Service Animals, and Assistance Animals",
"Residence Hall Rules &amp;amp; Regulations - Quiet Hours &amp;amp; Noise" =
  "GUL - Noise",
"Residence Hall Rules &amp;amp; Regulations - Recording Devices" =
  "GUL - Recording Devices",
"Residence Hall Rules &amp;amp; Regulations - Restrooms" =
  "GUL - Restrooms",
"Residence Hall Rules &amp;amp; Regulations - Sign Posting" =
  "GUL - Sign Posting",
"Residence Hall Rules &amp;amp; Regulations - Tobacco-Free Policy" =
  "GUL - Tobacco-Free Policy",
```

```
  "Residence Hall Rules &amp;amp; Regulations - Weapons &amp;amp; Fireworks" =
    "GUL - Weapons and Fireworks",
  "Residence Hall Rules &amp;amp; Regulations - Water Fights &amp;amp; Games" =
    "GUL - Water Fights and Games",
  "Residence Hall Rules &amp;amp; Regulations - Windows" = "GUL - Windows and Exits",
  "Residence Hall Rules &amp;amp; Regulations - Windows &amp;amp; Exits" =
    "GUL - Windows and Exits",
  "Unauthorized use of university key" =
    "Unauthorized Use of University Key",
  "University policies or rules" =
    "University Policies or Rules",
  "University policies or rules â€“ COVID-19 Safety Measures and Protocols" =
    "COVID-19 Safety",
  "University policies or rules – COVID-19 Safety Measures and Protocols" =
    "COVID-19 Safety",
  "University policies or rules \x96 COVID-19 Safety Measures and Protocols" =
    "COVID-19 Safety",
  "Harassment or Discrimination/Dating Violence" =
    "Harassment or Discrimination",
  "Harassment or Discrimination/Sexual/gender-based Harassment" =
    "Harassment or Discrimination",
  "Harassment or Discrimination/Sexual-gender based violence" =
    "Harassment or Discrimination",
  "Harassment or Discrimination/Stalking" =
    "Harassment or Discrimination",
  "Violation of federal, state, or local law" =
    "Violation of Federal, State, or Local Law",
  "Tobacco and Smoke Free Environment Policy" =
    "Tobacco and Smoking"
)
```

```
# Sanction Name Standardization ------------------------------------------

SANCTION_REPLACEMENTS <- c(
  "ABSENCE" =
    "Absence",
  "ACDISMISS" =
    "Academic Dismissal",
  "ACSUSP" =
    "Academic Suspension",
  "ADDENDUM" =
    "Addendum",
  "ADMS" =
    "Alcohol Decision Making Seminar",
  "ADP" =
    "Academic Probation",
  "ADR" =
    "Academic Reprimand",
  "ALDRPAPER" =
    "Alcohol and Drug Reflection Paper",
  "ANGERMGMT" =
    "Anger Management Course",
  "APOLOGY" =
    "Apology Letter",
  "ASEP" =
    "Alcohol Skills Education Program",
  "ASSESSMENT" =
    "Assessment with CAPS",
  "AUTOBIO_PAPER" =
    "Substance Autobiography Paper",
```

```
  "BASICS" =
    "Brief Alcohol Screening and Intervention for College Students",
  "BEHAVAGREE" =
    "Behavioral Agreement",
  "BEHAVIORMOD" =
    "Behavior Modification Course",
  "BULLETIN" =
    "Bulletin Board",
  "BULLYING" =
    "Bullying Course",
  "BYSTANDER" =
    "Bystander Intervention Seminar",
  "CANNABISCRS" =
    "The Cannabis Course",
  "CHNGDIR" =
    "Director Change of Information",
  "CHNGPERCENT" =
    "Reduced Grade for Course",
  "CLASSACCM" =
    "Classroom Accommodations",
  "COMMUNITY_SERVICE" =
    "Community Service",
  "CONSENT_WKSP" =
    "Consent Workshop",
  "CONTACT_PAPER" =
    "Contract Tracing Exercise and Reflection",
  "CPSRAPIDCONSULT" =
    "CAPS Rapid Access Consultation",
  "DMS" =
    "Decision Making Seminar",
```

```
"DRUGALCAWARE" =
  "Drug & Alcohol Awareness Course",
"DSEP" =
  "Drug Skills Education Program",
"EDPROGP" =
  "Plagiarism Module",
"EDU_SIGN" =
  "Educational Sign or Flier",
"EMPIM" =
  "Employment Interim Measure",
"EXPULSION" =
  "Dismissal",
"FAIL" =
  "Reduced Grade on Exam/Assignment",
"FIRESAFEED" =
  "Fire Safety Education",
"GRADECHNG" =
  "Failure in the Course",
"HEALTHY_REL_WKSP" =
  "Healthy Relationships Workshop",
"HEALTH_PAPER" =
  "Health Policy Video and Reaction Paper",
"HMW" =
  "Healthy Masculinity Workshop",
"INTERIMSUSP" =
  "Interim Suspension",
"LACROOM" =
  "Lactation Room Access",
"LIFESKILLS" =
  "Life Skills Course",
```

```
"LOSSPRIV" =
  "Loss of Privileges",
"MISSASGN" =
  "Missed Assignment",
"MOVETOHSNG" =
  "Move into Housing with Financial Assistance",
"MOVETOHSNG_NOASST" =
  "Move into Housing without Financial Assistance",
"NCO_OEO" =
  "Mutual No-Contact Order (OEO)",
"NGHTRIDE" =
  "NightRide Priority Ride",
"NOCONTACT" =
  "No-Contact Directive",
"ORG_RESTORE" =
  "Student Organization Restorative Action Plan",
"PACE" =
  "PACE Workshop",
"PARENTNOTIFY" =
  "Parent/Guardian Notification",
"PARKING" =
  "Parking with Financial Assistance",
"PARKING_NOASST" =
  "Parking without Financial Assistance",
"PFL" =
  "Substance Abuse Education for Alcohol",
"PFLDRUG" =
  "Drug Sanction Course",
"REDEXPUL" =
  "Housing Termination and Permanent Loss of Privileges",
```

```
  "REDSUSP" =
    "Housing Termination and Temporary Loss of Privileges",
  "REDTRANSFER" =
    "Housing Assignment Relocation",
  "REFERRAL" =
    "Referral",
  "REFLECTCONT" =
    "Reflection Paper",
  "REFLECTOPEN" =
    "Reflection Paper - Open-Ended",
  "REFLECTPAP" =
    "Reflection Paper - Preparation",
  "REFLECTPRE" =
    "Reflection Paper - Pre-Contemplation",
  "REFLGTUL" =
    "GUL Reflection",
  "REFMEET" =
    "Reflection Meeting",
  "REFUNDAPP" =
    "Tuition Refund Application",
  "REGREIMBRSE" =
    "Registrar Grade Removal with Reimbursement",
  "RELOCATION" =
    "Housing Relocation",
  "REPRIMAND" =
    "Reprimand",
  "RESTITUTION" =
    "Restitution",
  "RESTRICTION" =
    "Ban From Location",
```

```
  "RESUBASSIGN" =
    "Retest/Re-Submission of the Assignment/Exam",
  "RP_PASSIVE" =
    "Reflection Paper - Passive Participation",
  "SCHLLTR" =
    "Scholarship Letter",
  "SECURITYASST" =
    "Security Assistance",
  "SOBEREXP" =
    "Sober Experience Calendar",
  "TAIALCOHOL" =
    "Think About It: Alcohol",
  "TAIDRUGS" =
    "Think About It: Drugs",
  "TEDTALK" =
    "TED Talk Reflection",
  "TERMINATION" =
    "Housing Removal",
  "THEFTAWARE" =
    "Theft Awareness Course",
  "TOBACCO" =
    "Tobacco Awareness Course",
  "VAPING" =
    "Vaping Awareness Course",
  "DEFSUSHP" =
    "Deferred Suspension of Housing Privileges",
  "DP" =
    "Disciplinary Probation",
  "SUSPENSION" =
    "Suspension",
```

```
  "ADDITIONAL" =
    "Additional Sanctions or Stipulations"
)


# College Abbreviations ------------------------------------------------

COLLEGE_REPLACEMENTS <- c(
  'Colege of Dsn, Arch, Art & Pln' = 'DAAP',
  'Lindner College of Business' = 'LCOB',
  'Col of Arts & Science' = 'CAS',
  'College of Ed, CJ, & HS' = 'CECH',
  'Blue Ash College' = 'UCBA',
  'College of Nursing' = 'CON',
  'College of Eng & Appl Sci' = 'CEAS',
  'University of Cincinnati' = 'UC',
  'Clermont College' = 'UCCC',
  'College of Allied Health Sci' = 'CAHS',
  'College of Medicine' = 'COM',
  'College Conservatory of Music' = 'CCM',
  'UC International Pathways' = 'UCIP',
  'Adult Learning Center' = 'ALC',
  'James L. Winkle Coll of Pharm' = 'COP'
)


# Academic Calendar Pause Periods --------------------------------------
# Breaks, holidays, and other periods to exclude from timeline calculations

OMIT_RANGES <- list(
```

```
# AY1718
c("2017-08-06", "2017-08-20"), # summer to fall transition
c("2017-09-04", "2017-09-04"), # labor day
c("2017-11-10", "2017-11-10"), # veterans day
c("2017-11-22", "2017-11-26"), # thanksgiving holiday
c("2017-12-10", "2018-01-07"), # semester break
c("2018-01-15", "2018-01-15"), # MLK Day
c("2018-03-12", "2018-03-16"), # spring break
c("2018-04-27", "2018-05-06"), # spring to summer transition
c("2018-05-28", "2018-05-28"), # memorial day
c("2018-07-04", "2018-07-04"), # independence day

# AY1819
c("2018-08-06", "2018-08-26"), # summer to fall transition
c("2018-09-04", "2018-09-04"), # labor day
c("2018-10-11", "2018-10-12"), # reading days
c("2018-11-12", "2018-11-12"), # veterans day
c("2018-11-22", "2018-11-25"), # thanksgiving holiday
c("2018-12-10", "2019-01-13"), # semester break
c("2019-01-21", "2019-01-21"), # MLK Day
c("2019-03-18", "2019-03-24"), # spring break
c("2019-04-27", "2019-05-12"), # spring to summer transition
c("2019-05-27", "2019-05-27"), # memorial day
c("2019-07-04", "2019-07-04"), # independence day

# AY1920
c("2019-08-11", "2019-08-25"), # summer to fall transition
c("2019-09-02", "2019-09-02"), # labor day
c("2019-10-10", "2019-10-11"), # reading days
c("2019-11-11", "2019-11-11"), # veterans day
```

```
c("2019-11-28", "2019-12-01"), # thanksgiving holiday
c("2019-12-09", "2020-01-12"), # semester break
c("2020-01-20", "2020-01-20"), # MLK Day
c("2020-03-16", "2020-03-25"), # spring break
c("2020-04-26", "2020-05-10"), # spring to summer transition
c("2020-05-25", "2020-05-25"), # memorial day
c("2020-07-03", "2020-07-03"), # independence day

# AY2021
c("2020-08-09", "2020-08-23"), # summer to fall transition
c("2020-09-07", "2020-09-07"), # labor day
c("2020-11-11", "2020-11-11"), # veterans day
c("2020-11-26", "2020-11-29"), # thanksgiving
c("2020-12-03", "2021-01-10"), # semester break
c("2021-01-18", "2021-01-18"), # MLK Day
c("2021-04-24", "2021-05-09"), # spring to summer transition
c("2021-05-31", "2021-05-31"), # memorial day
c("2021-07-05", "2021-07-05"), # independence day

# AY2122
c("2021-08-10", "2021-08-22"), # summer to fall transition
c("2021-09-06", "2021-09-06"), # labor day
c("2021-10-11", "2021-10-12"), # reading days
c("2021-11-11", "2021-11-11"), # veterans day
c("2021-11-25", "2021-11-28"), # thanksgiving holiday
c("2021-12-05", "2022-01-09"), # semester break
c("2022-01-17", "2022-01-17"), # MLK Day
c("2022-03-14", "2022-03-20"), # spring break
c("2022-04-22", "2022-05-08"), # spring to summer transition
c("2022-05-30", "2022-05-30"), # memorial day
```

```
    c("2022-06-20", "2022-06-20"), # juneteenth
    c("2022-07-04", "2022-07-04"), # independence day

    # AY2223
    c("2022-08-07", "2022-08-21"), # summer to fall transition
    c("2022-09-05", "2022-09-05"), # labor day
    c("2022-10-10", "2022-10-10"), # reading day
    c("2022-11-08", "2022-11-08"), # reading day
    c("2022-11-11", "2022-11-11"), # veterans day
    c("2022-11-24", "2022-11-27"), # thanksgiving holiday
    c("2022-12-04", "2023-01-08"), # semester break
    c("2023-01-16", "2023-01-16"), # MLK Day
    c("2023-03-13", "2023-03-19"), # spring break
    c("2023-04-22", "2023-05-07"), # spring to summer transition
    c("2023-05-29", "2023-05-29"), # memorial day
    c("2023-06-19", "2023-06-19"), # juneteenth
    c("2023-07-04", "2023-07-04"), # independence day

    # AY2324
    c("2023-08-06", "2023-08-20"), # summer to fall transition
    c("2023-09-04", "2023-09-04"), # labor day
    c("2023-10-09", "2023-10-09"), # reading day
    c("2023-11-07", "2023-11-07"), # reading day
    c("2023-11-10", "2023-11-10"), # veterans day
    c("2023-11-23", "2023-11-26"), # thanksgiving holiday
    c("2023-12-03", "2024-01-07"), # semester break
    c("2024-01-15", "2024-01-15"), # MLK Day
    c("2024-03-11", "2024-03-17"), # spring break
    c("2024-04-20", "2024-05-05"), # spring to summer transition
    c("2024-05-27", "2024-05-27"), # memorial day
```

```
    c("2024-06-19", "2024-06-19"), # juneteenth
    c("2024-07-04", "2024-07-04"), # independence day

    # AY2425
    c("2024-08-04", "2024-08-25"), # summer to fall transition
    c("2024-09-02", "2024-09-02"), # labor day
    c("2024-10-11", "2024-10-11"), # reading day
    c("2024-11-05", "2024-11-05"), # reading day
    c("2024-11-11", "2024-11-11"), # veterans day
    c("2024-11-28", "2024-12-01"), # thanksgiving holiday
    c("2024-12-08", "2025-01-12"), # semester break
    c("2025-01-20", "2025-01-20"), # MLK Day
    c("2025-01-13", "2025-02-14"), # Datafeed error schedules
    c("2025-03-17", "2025-03-23"), # spring break
    c("2025-04-26", "2025-05-11"), # spring to summer transition
    c("2025-05-26", "2025-05-26"), # memorial day
    c("2025-06-19", "2025-06-19"), # juneteenth
    c("2025-07-04", "2025-07-04"), # independence day

    # AY2526
    c("2025-08-10", "2025-08-24"), # summer to fall transition
    c("2025-09-01", "2025-09-01"), # labor day
    c("2025-10-09", "2025-10-10"), # reading day
    c("2025-11-11", "2025-11-11"), # veterans day
    c("2025-11-26", "2025-11-28"), # thanksgiving holiday
    c("2025-12-06", "2026-01-11"), # semester break
    c("2026-01-19", "2026-01-19"), # MLK Day
    c("2026-03-16", "2026-03-20"), # spring break
    c("2026-05-01", "2026-05-10"), # spring to summer transition
    c("2026-05-25", "2026-05-25"), # memorial day
```

```
  c("2026-06-19", "2026-06-19"), # juneteenth
  c("2026-07-03", "2026-07-03")  # independence day
)
```