

SRMS Data Pipeline Guide

(Akari, Brightspace, Banner)

Technological University Dublin

Paul Doyle

Dean, Faculty of Engineering, Built Environment and Apprenticeships

October 27, 2025

Contents

| | |
|--|-----------|
| Scope | 3 |
| 1 Directory Structure | 4 |
| 2 Akari Cleaner (src/akari_clean.py) | 5 |
| 3 R27 Raw Data Consistency Test (src/CheckStudentCountConsistency.py) | 7 |
| 4 R27 Cleaner (src/r27_clean.py) | 10 |
| 5 R27 Duplicate Remover (src/r27_dedupe.py) | 12 |
| 6 Programmes with Dummy CRN Cleaner (src/programmes_dummy_crn_clean.py) | 14 |
| 7 R27 Augmentation with Dummy CRN (src/r27_augment_with_dummy_crn.py) | 16 |
| 8 CRN2Use Cleaner (src/crn2use_clean.py) | 18 |
| 9 R27 Flagging by CRN2USE (src/r27_flag_crn2use.py) | 19 |
| 10 Brightspace Cleaner (src/brightspace_clean.py) | 20 |
| 11 R27 Update with Brightspace Student Counts | 22 |
| 12 Viewer Builders | 24 |
| 13 Akari Viewer Builder (src/build_programme_viewer.py) | 25 |
| 14 R27 Viewer Builder (src/build_R27_viewer.py) | 26 |
| 14.1 General Learner Record Viewer Enhancement | 26 |
| 14.1.1 Design Objectives | 27 |
| 14.1.2 Functional Overview | 27 |
| 14.1.3 Interface Behaviour | 27 |
| 14.1.4 Implementation Notes | 27 |
| 14.1.5 Output | 28 |
| 15 Operational Checklist | 29 |
| 16 Troubleshooting | 30 |

Scope

This guide documents the full SRMS pipeline from raw inputs to HTML viewers:

- Akari cleaner: `src/akari_clean.py`
- R27 cleaner (S1/S2): `src/r27_clean.py`
- R27 duplicate remover: `src/r27_dedupe.py`
- Programmes with Dummy CRN cleaner: `src/programmes_dummy_crn_clean.py`
- R27 augmentation with Dummy CRN: `src/r27_augment_with_dummy_crn.py`
- CRN2Use cleaner (Excel → per-term CSVs): `src/crn2use_clean.py`
- R27 flagging by CRN2USE: `src/r27_flag_crn2use.py`
- Akari viewer: `src/viewer_akari_build.py`
- Brightspace cleaner: `src/brightspace_clean.py`
- R27 update with Brightspace Student Counts: `src/r27_update_with_brightspace.py`
- R27 viewers: `src/viewer_r27_build.py`
- End-to-end runner: `src/pipeline_run.sh`

Chapter 1

Directory Structure

```
SRMS-Dashboard/  
  DataSources/  
    Akari-Programme and Module data.xlsx  
    R27_Expanded_202510.csv  
    R27_Expanded_202520.csv  
    Programmes_with_Dummy_CRN.csv  
    CRN2Use.xlsx # sheets: 202510, 202520, 202530 (others ignored)  
  cleancsv/  
    akari_clean.csv  
    r27_202510_clean.csv  
    r27_202520_clean.csv  
    programmes_dummy_crn_clean.csv  
    programmes_dummy_crn_conflicts.csv # only if conflicts  
    crn2use_202510.csv  
    crn2use_202520.csv  
    crn2use_202530.csv # if present  
    (same r27_* files updated in-place with extra columns)  
  html/  
    viewer_akari_by_school.html  
    viewer_r27_202510.html  
    viewer_r27_202520.html  
  src/  
    brightspace_clean.py  
    r27_update_with_brightspace.py  
    akari_clean.py  
    r27_clean.py  
    r27_dedupe.py  
    programmes_dummy_crn_clean.py  
    r27_augment_with_dummy_crn.py  
    crn2use_clean.py  
    r27_flag_crn2use.py  
    viewer_akari_build.py  
    viewer_r27_build.py  
    pipeline_run.sh
```

Chapter 2

Akari Cleaner (src/akari_clean.py)

Input/Output

- Input: ../DataSources/Akari-Programme and Module data.xlsx
- Output: ../cleancsv/akari_clean.csv

Selected Columns & Final Order (exact)

| Output Column | Source / Notes |
|-------------------------|---|
| TU Programme Code | From sheet column of the same meaning |
| Year | From sheet “Year” (string-preserved) |
| Semester | Normalised to one of: 1, 1&2, 2 |
| Code | Module code (as-is) |
| Title | <i>Module</i> name |
| Delivery Type | As-is |
| Credits | As-is (kept as string; numeric on client) |
| Programme Title | As-is |
| School Responsible | Case-normalised to avoid duplicates (e.g. “of” vs “Of”) |
| Full Time/Part Time | As-is |
| Campus | As-is |
| NFQ Level | As-is |
| Programme Type | As-is |
| Programme Delivery Mode | As-is |

Normalisation Rules

- Header matching: tolerant (case/spacing/punctuation).
- Semester: mapped to 1, 1&2, 2.
- All fields read as text to preserve codes; UTF-8 BOM on output.

Run

```
cd src  
python Akari-CSV-Clean.py
```

Chapter 3

R27 Raw Data Consistency Test (src/CheckStudentCountConsistency.py)

Purpose

Before any data cleaning or transformation is performed, the R27 raw file (R27_Expanded_202510.csv) is validated to ensure internal consistency of student registration data. This step identifies modules where STUDENT_COUNT values are not consistent across records that share the same key identifiers.

Test Logic

The script verifies that for each unique combination of:

- Programme Code
- CRN
- SSRRMAJ_MAJR_CODE
- Module (or Module Name)

the recorded STUDENT_COUNT remains the same across all rows. If different student counts exist for the same key, the record group is flagged as inconsistent.

Script Implementation

```
import pandas as pd
from pathlib import Path
import re

DATA_DIR = Path("../DataSources")
INPUT_FILE = DATA_DIR / "R27_Expanded_202510.csv"
OUTPUT_FILE = DATA_DIR / "R27_inconsistent_student_counts.csv"

df = pd.read_csv(INPUT_FILE, encoding='utf-8-sig', low_memory=False)
df.columns = [re.sub(r'^A-Za-z0-9', '', c).lower() for c in df.columns]
```



```

# Identify relevant columns
programme_col = 'programmecode'
crn_col = 'crn'
major_col = 'ssrrmajmajrcode'
module_col = 'module'
student_col = 'studentcount'

df[student_col] = pd.to_numeric(df[student_col], errors='coerce')

# Group and compare counts
key_cols = [programme_col, crn_col, major_col, module_col]
summary = (df.groupby(key_cols)[student_col]
            .agg(['count', 'nunique', 'min', 'max'])
            .reset_index()
            .rename(columns={'count': 'RecordCount',
                              'nunique': 'DistinctStudentCounts',
                              'min': 'MinStudentCount',
                              'max': 'MaxStudentCount'}))

# Filter inconsistencies
inconsistent = summary[summary['DistinctStudentCounts'] > 1]
inconsistent.to_csv(OUTPUT_FILE, index=False, encoding='utf-8-sig')
print(f"Inconsistent combinations written to: {OUTPUT_FILE}")

```

Results

The test was executed on the unprocessed dataset before running any cleaning operations. The output confirmed full consistency of student registration counts across all module records.

```

(base) paul@soc-MacMini-PD src % python CheckStudentCountConsistency.py
Loading file: ../DataSources/R27_Expanded_202510.csv
Detected columns:
  Programme: programmecode
    CRN: crn
   Major: ssrrmajmajrcode
  Module: module
Student Count: studentcount

Consistent module records: 21,729
Inconsistent module records: 0
Inconsistent combinations written to: ../DataSources/R27_inconsistent_student_counts.csv

Sample inconsistent combinations:
Empty DataFrame
Columns: [programmecode, crn, ssrrmajmajrcode, module, RecordCount,
         DistinctStudentCounts, MinStudentCount, MaxStudentCount]
Index: []

```

Conclusion

The R27 raw file for term 202510 demonstrated complete internal consistency. No cases were found where the `STUDENT_COUNT` differed for records sharing the same `Programme Code`, `CRN`, `SSRRMAJ_MAJR_CODE`, and `Module`. This confirms that registration data is stable at the raw source level and subsequent cleaning processes can safely assume one consistent student count per module delivery.

Chapter 4

R27 Cleaner (src/r27_clean.py)

Inputs/Outputs

- Inputs: ../DataSources/R27_Expanded_202510.csv, ../DataSources/R27_Expanded_202520.csv
- Outputs: ../cleancsv/r27_202510_clean.csv, ../cleancsv/r27_202520_clean.csv

Run

```
cd src
python r27_clean.py
```

Selected Columns & Final Order

| Output Column | Source / Transform |
|---------------------|--|
| TU Programme Code | Programme_Code |
| Year | Extracted from SMRARUL_AREA: last digits; fallback: F/P/FP digits (e.g. FP3 → 3) |
| Semester | From Part_of_Term mapped: 1 → Semester 1; 2 → Semester 2; FY/Y → Semester 1 & 2 |
| Code | SMRARUL_SUBJ_CODE + space + SMRARUL_CRSE_NUMB_LOW |
| Title | Module (module name) |
| Programme Title | Programme_Desc (programme name) |
| Delivery Type | SMBARUL_KEY_RULE |
| Credits | Credits (kept as string) |
| Campus | Campus |
| School Responsible | Mapped from Dept (see mapping) |
| SMRPRLE_LEVL_CODE | As-is |
| Term_Code | As-is |
| CRN | CRN |
| Full Time/Part Time | SSRRMAJ_MAJR_CODE |
| STUDENT_COUNT | As-is |

| | |
|-----------|-----------|
| StaffID | ID |
| FirstName | FirstName |
| LastName | LastName |

Dept → School Responsible Mapping

The cleaner applies a fixed mapping. Key examples:

| Dept | School Responsible |
|------|--|
| SABE | School of Architecture, Building and Environment |
| SSCI | School of Surveying & Construction Innovation |
| SCSC | School of Computer Science |
| SEEE | School of Electrical & Electronic Engineering |
| GRDR | Graduate Research |
| CONS | Conservatoire |
| GBUS | Graduate Business School |
| SFEH | School of Food Science & Environmental Health |

... full list embedded in the script; unknown codes pass through as their raw Dept value and are reported in a warning.

Header Matching & Normalisation

- Tolerant header matching: case/underscores/punctuation (e.g. `programme code` ~ `Programme.Code`).
- Whitespace collapsed; values trimmed.
- Diagnostics printed: row/column counts; unmapped Dept codes; count of rows missing Module.

Chapter 5

R27 Duplicate Remover (src/r27_dedupe.py)

Purpose

Remove duplicate rows in the cleaned R27 outputs. Duplicates are determined after normalising whitespace and ignoring staff-identifying fields (and optionally Delivery Type), as configured in the script.

Inputs/Outputs

- Inputs/Outputs (overwritten in place): ../cleancsv/r27_202510_clean.csv, ../cleancsv/r27_202520

Run

```
cd src
python r27_dedupe.py
```

Process

1. Loads each cleaned CSV using **pandas**.
2. Identifies fully identical rows (same values across all columns).
3. Removes duplicates and overwrites the original file.
4. Reports the total number of rows before and after cleaning.

Example Output

```
[INFO] Processing: R27_Expanded_202510_clean.csv
Total rows: 23891
Duplicates removed: 121
Final rows: 23770
File overwritten: ../cleancsv/R27_Expanded_202510_clean.csv
```

```
[INFO] Processing: R27_Expanded_202520_clean.csv
Total rows: 25032
Duplicates removed: 98
Final rows: 24934
File overwritten: ../cleancsv/R27_Expanded_202520_clean.csv

[INFO] Duplicate removal complete.
```

Chapter 6

Programmes with Dummy CRN Cleaner

(src/programmes_dummy_crn_clean.py)

Purpose

Normalise the mapping of TU Programme Code → CRNs for 202510/202520 from the CSV export.

Inputs/Outputs

- Input: ../DataSources/Programmes_with_Dummy_CRN.csv
- Outputs: ../cleancsv/programmes_dummy_crn_clean.csv
../cleancsv/programmes_dummy_crn_conflicts.csv (only if conflicts)

Run

```
cd src
python programmes_dummy_crn_clean.py
```

Process

1. Read CSV as text to preserve codes verbatim.
2. Extract `programme_code` using regex `(TU\d{3,4}[A-Z]?)` from the `Programme` column; if the cell is itself a TU code, use it directly.
3. Trim and normalise whitespace in CRN columns (202510/202520).
4. Group by `programme_code`:
 - Output one row per programme code; choose the *mode* CRN per term (deterministic tie-break) for stability.
 - If more than one distinct CRN exists for a programme/term, record the case in the conflicts CSV.

5. Write the cleaned CSV (UTF-8 with BOM).

Notes

- If the source remains in Excel format, export to CSV first to ensure consistent parsing.
- The conflicts CSV is advisory; if a hard failure is preferred on CRN clashes, switch the script to raise on `nunique > 1`.

Chapter 7

R27 Augmentation with Dummy CRN

(src/r27_augment_with_dummy_crn.py)

Purpose

Left-join the Dummy CRN mapping by TU Programme Code and add: DummyCRN_202510 to r27_202510_clean.csv and DummyCRN_202520 to r27_202520_clean.csv.

Inputs/Outputs

- Inputs: ../cleancsv/r27_202510_clean.csv, ../cleancsv/r27_202520_clean.csv, ../cleancsv/programmes_with_dummy_crn_clean.csv
- Outputs: same R27 files overwritten in-place with the new columns.

Run

```
cd src
python r27_augment_with_dummy_crn.py
```

Process

1. Read the cleaned R27 CSV and the Programmes_with_Dummy_CRN_clean.csv mapping.
2. Normalise the TU Programme Code column in both datasets (trim, uppercase, remove stray BOMs).
3. Perform a left join on TU Programme Code.
4. Insert the appropriate dummy CRN column:
 - DummyCRN_202510 for the 202510 file.
 - DummyCRN_202520 for the 202520 file.
5. Overwrite the original cleaned R27 files.

Example Output

```
[INFO] Augmenting: R27_Expanded_202510_clean.csv
Added column: DummyCRN_202510
Rows with mapped CRN: 1248 / 23770
File overwritten: ../cleancsv/R27_Expanded_202510_clean.csv

[INFO] Augmenting: R27_Expanded_202520_clean.csv
Added column: DummyCRN_202520
Rows with mapped CRN: 1274 / 24934
File overwritten: ../cleancsv/R27_Expanded_202520_clean.csv
[INFO] Augmentation complete.
```

Notes

- Matching is case-insensitive and whitespace-normalised.
- The `DummyCRN_*` columns remain empty where the TU Programme Code is not found in the mapping.
- This step must be run *after* both the R27 and the Programmes-with-Dummy-CRN cleaners.

Chapter 8

CRN2Use Cleaner (src/crn2use_clean.py)

Purpose

Ingest the Excel `CRN2Use.xlsx` and output one CSV per required sheet (terms only).

Inputs/Outputs

- Input: `../DataSources/CRN2Use.xlsx` (sheets processed: 202510, 202520, 202530; e.g. sheet index is ignored)
- Outputs: `../cleancsv/crn2use_202510.csv`, `../cleancsv/crn2use_202520.csv`, `../cleancsv/crn2u`

Output Columns (exact)

- `module code` \leftarrow column C&D (or built from `SSBSECT.SUBJ_CODE` + `SSBSECT.CRSE_NUMB`)
- `Programme Code` \leftarrow `SSRRPRG_PROGRAM`
- `CRN2Use` \leftarrow CRN to Use

Run

```
cd src
python crn2use_clean.py
```

Chapter 9

R27 Flagging by CRN2USE (src/r27_flag_crn2use.py)

Purpose

Mark rows in each R27 cleaned file where the CRN appears in the corresponding `crn2use_{term}.csv`. Adds a new column `CRN2USE` equal to the row's CRN when present in the list; blank otherwise.

Inputs/Outputs

- Inputs: `../cleancsv/r27_202510_clean.csv`, `../cleancsv/r27_202520_clean.csv`, `../cleancsv/crn2use_202510.csv`, `../cleancsv/crn2use_202520.csv`
- Outputs: same R27 files overwritten in-place, with an added `CRN2USE` column per row.

Run

```
cd src
python r27_flag_crn2use.py
```

Chapter 10

Brightspace Cleaner (src/brightspace_clean.py)

Purpose

Ingest Brightspace enrolment data and produce a minimal, normalised CSV for downstream joins/validation.

Inputs/Outputs

- Input (Excel): ../DataSources/BrightspaceDump.xlsx (sheet: All COs with roles)
- Output (CSV, UTF-8 BOM): ../cleancsv/brightspace_brightspace_dump_clean.csv

Output Columns (exact order)

| Column | Source / Transform |
|---------------|---|
| Code | From CourseCode: token 1 (title-cased) + space + token 2. Example: DATA-H1010-... → Data H1010. |
| CRN | From CourseCode: token 3, preserved as string; must match <code>\d{5}</code> . |
| Semester | From CourseCode: token 6 mapped {202510→Semester 1, 202520→Semester 2, 202530→Semester 3}; otherwise blank. |
| StudentCount | From sheet column StudentCount; coerced to numeric (NaN on non-numeric). |
| LecturerCount | From sheet column LecturerCount; coerced to numeric (NaN on non-numeric). |

Parsing Rules (CourseCode)

- Split on -; expected 6 tokens (strict logical order): SUBJECT-MODULE-CRN-PROGRAMME-DELIVERY-TERMCODE
- If #tokens < 6, pad with None. If > 6, merge any extra tokens into the last position (term token) to avoid index errors.

- **Code** := SUBJECT (title-cased) + space + MODULE.
- **CRN** := token 3 (kept as string).
- **Semester** := map final token {202510,202520,202530} to {Semester 1, 2, 3}; no mapping \Rightarrow blank.

Row Filters (applied in this order)

1. **StudentCount**: keep rows with $0 < \text{StudentCount} \leq 1000$.
2. **Identifiers**: drop rows with missing **Code** or missing **CRN**.
3. **CRN format**: keep rows where **CRN** matches `\d{5}` (five numeric digits only); all others dropped.

Header Tolerance

- **CourseCode** is located via tolerant header matching (case/punctuation stripped). If not found \Rightarrow hard error.
- **StudentCount** and **LecturerCount** must be present; absence \Rightarrow hard error.

Runtime Summary (stdout)

On each run the script prints:

- records loaded from sheet;
- records kept/removed by the **StudentCount** filter;
- rows dropped for missing/invalid **Code**/**CRN** (including non-5-digit **CRNs**);
- final row count and output path.

Run

```
cd src
python brightspace_clean.py
```

Failure Modes & Remedies

- **File not found**: verify `../DataSources/BrightspaceDump.xlsx` exists and sheet name is exactly `All COs with roles`.
- **Missing columns**: ensure **CourseCode**, **StudentCount**, **LecturerCount** exist in the sheet (header spelling/spacing may vary; **CourseCode** is matched tolerantly, counts are required).
- **Malformed CourseCode**: rows with insufficient tokens may yield missing **Code**/**CRN** or non-mapped **Semester**; such rows are dropped by the identifier/**CRN** filters.
- **Term code not mapped**: if the last token is not one of 202510/202520/202530, **Semester** is blank (row still included if other criteria pass).

Chapter 11

R27 Update with Brightspace Student Counts

Purpose

Populate a new column **BrightSpace_StudentCount** in the R27 cleaned CSVs by joining Brightspace counts on **Code+CRN**, using the appropriate Brightspace semester partition.

Inputs/Outputs

- Inputs:
 - `../cleancsv/r27_202510_clean.csv` (target: Semester 1)
 - `../cleancsv/r27_202520_clean.csv` (target: Semester 2)
 - `../cleancsv/brightspace_brightspace_dump_clean.csv`
- Outputs: R27 files *updated in place* with a new column **BrightSpace_StudentCount**.

Join Logic (exact)

1. Brightspace rows are partitioned by **Semester**:
 - **Semester 1** \Rightarrow used to update `r27_202510_clean.csv`.
 - **Semester 2** \Rightarrow used to update `r27_202520_clean.csv`.
2. Match key: **Code + CRN**. Normalisation of **Code** on both sides:
 - strip leading/trailing whitespace and BOM;
 - collapse internal whitespace to single spaces;
 - uppercase.
3. Only Brightspace rows with **CRN** matching `^\d{5}$` are retained (consistent with the Brightspace cleaner).
4. If multiple Brightspace rows map to the same (**Code**, **CRN**) within a semester, the updater takes the **max StudentCount**.
5. For R27 rows with no Brightspace match, **BrightSpace_StudentCount** is left blank.

Column Added

| Column | Notes |
|--------------------------|---|
| BrightSpace.StudentCount | Integer from Brightspace StudentCount (per matched Code+CRN); blank if no match. |

Run

```
cd src
python r27_update_with_brightspace.py
```

Failure Modes & Checks

- Missing **Code/CRN** columns in R27 files \Rightarrow hard error.
- Missing or empty Brightspace CSV \Rightarrow no updates applied; script logs a warning per semester.
- Non-5-digit CRNs in Brightspace are ignored by design.

Chapter 12

Viewer Builders

Akari Viewer (src/viewer_akari_build.py)

- Input: ../cleancsv/akari_clean.csv
- Output: ../html/viewer_akari_by_school.html

R27 Viewers (src/viewer_r27_build.py)

- Inputs: ../cleancsv/r27_202510_clean.csv, ../cleancsv/r27_202520_clean.csv
- Outputs: ../html/viewer_r27_202510.html, ../html/viewer_r27_202520.html

Run

```
cd src
python viewer_akari_build.py
python viewer_r27_build.py
```

Chapter 13

Akari Viewer Builder (src/build_programme_viewer.py)

Purpose

Generate an interactive, self-contained HTML (`../html/programme_viewer_by_school.html`) for exploring Akari data by:

1. School Responsible
2. Programme (Title & TU Programme Code)
3. Year

It displays modules grouped by semester (1, 1&2, 2) and totals credits per group.

Input/Output

- Input: `../cleancsv/Akari-ProgrammeAndModuleData.csv`
- Output: `../html/programme_viewer_by_school.html`

Notes

- Data is embedded (no runtime file access).
- Credits summed client-side (parsed as floats).

Run

```
cd src
python build_programme_viewer.py
```

Chapter 14

R27 Viewer Builder (src/build_R27_viewer.py)

Purpose

Generate two separate self-contained HTML viewers using only R27 cleaned CSVs:

- ../html/r27_viewer_202510.html (Semester 1 data)
- ../html/r27_viewer_202520.html (Semester 2 data)

Filters mirror the Akari viewer: School → Programme Code → Year; optional: Campus, Full Time/Part Time. Tables include columns: **Code**, **CRN**, **Title**, **Credits**, **Delivery Type**, **Students**. Semester credit totals are shown.

Inputs/Outputs

- Inputs:
 - ../cleancsv/R27_Expanded_202510_clean.csv
 - ../cleancsv/R27_Expanded_202520_clean.csv
- Outputs:
 - ../html/r27_viewer_202510.html
 - ../html/r27_viewer_202520.html

Run

```
cd src
python build_R27_viewer.py
```

14.1 General Learner Record Viewer Enhancement

The General Learner Record (GLR) Viewer has been updated to provide a simplified, unified interface for reviewing programme-level data across academic terms. The new implementation removes multiple selection filters in favour of a single, robust school-level filter and introduces a comparative visualisation of the two academic terms.

14.1.1 Design Objectives

The objective of this update was to:

- Simplify the user interaction model by retaining only one filter (**School Responsible**).
- Enable side-by-side comparison of **Term 202510** and **Term 202520**.
- Provide automatic summary statistics, including per-status counts and overall totals for each term.
- Maintain static portability (no server-side code required) while ensuring responsive and interactive functionality via embedded JavaScript.

14.1.2 Functional Overview

The viewer is generated from two cleaned data sources:

- `glr_term1_clean.csv` (Term 202510)
- `glr_term2_clean.csv` (Term 202520)

Each record represents a unique **Programme Code** and includes:

- The associated **School Responsible**
- The corresponding **DUMMY_CRN** for each term
- Six status category counts: **BLANK**, **EL**, **ES**, **NS**, **SI**, and **SW**

14.1.3 Interface Behaviour

- The **School** dropdown allows selection of either a specific school or all schools combined.
- Upon selection, the viewer dynamically displays two tables:
 - **Term 202510** (left column)
 - **Term 202520** (right column)
- Each table lists programme codes, their schools, dummy CRNs, and the six status counts.
- The summary panel above the tables displays:
 - The number of programmes currently shown.
 - A per-status count for both terms.
 - A new **TOTAL** row providing the overall count across all statuses per term.

14.1.4 Implementation Notes

- The viewer is built entirely from static assets using **Bootstrap 5** and embedded JavaScript.
- Data is pre-aggregated and converted to JSON within the build process using the Python script `build_glr_viewer.py`.
- No external dependencies or network access are required for rendering.

14.1.5 Output

The resulting static HTML file is written to:

```
../html/viewer_glr_by_school.html
```

It provides a permanent, portable reference for GLR data, allowing comparative inspection between terms without requiring database connectivity or manual filtering.

Chapter 15

Operational Checklist

1. Place sources in `DataSources/`:

- Akari Excel
- R27 raw CSVs: `R27_Expanded_202510.csv`, `R27_Expanded_202520.csv`
- `Programmes_with_Dummy_CRN.csv`
- `CRN2Use.xlsx` (tabs: 202510/202520/202530; others ignored)

2. Run the end-to-end pipeline:

```
cd src
./pipeline_run.sh
```

3. Outputs:

- Clean CSVs in `cleancsv/` (R27 files updated in-place with new columns).
- Viewers in `html/`.

4. The pipeline stages (in order) include:

- (a) Akari cleaner
- (b) R27 cleaner
- (c) R27 duplicate remover
- (d) Programmes with Dummy CRN cleaner
- (e) Brightspace cleaner
- (f) R27 augmentation with Dummy CRN
- (g) CRN2Use cleaner
- (h) R27 flagging by CRN2USE
- (i) **R27 update with Brightspace Student Counts** (*new; before viewers*)
- (j) Build Akari viewer
- (k) Build R27 viewers

Chapter 16

Troubleshooting

- **Missing columns:** Cleaners create blank columns if a source column is not found (tolerant header matching may still fail if columns are renamed drastically upstream).
- **Wrong semester:** Ensure `Part_of_Term` values are one of 1, 2, FY/Y. Other values pass through verbatim and may appear as “Other” in viewers.
- **Year extraction:** If `SMRARUL_AREA` lacks trailing digits and no `F/P/FP;digits;` pattern is present, Year will be blank.
- **Dept mapping:** Unknown `Dept` codes are left as-is in “School Responsible” and reported in cleaner output.
- **CSV encoding:** All outputs are UTF-8 with BOM for compatibility with Excel.

Chapter 17

Questions and Clarifications

Question 1: Meaning of Part of Term Codes in R27 Data

Context: The R27 raw export includes a field called `Part of Term`, typically containing one of the values:

- 1
- 2
- FY

In the current dataset (`R27_Expanded_202510.csv`), all records are for Semester 1. However, within this file, the `Part of Term` field still includes a mix of values such as 1, 2, and FY.

Current Handling in the Cleaning Process:

- 1 → mapped to `Semester 1`
- 2 → mapped to `Semester 2`
- FY (and Y, 1&2, 12) → mapped to `Semester 1 & 2`

This logic assumes that:

1. A value of 1 indicates a Semester 1 delivery.
2. A value of 2 indicates a Semester 2 delivery.
3. A value of FY indicates a full-year delivery (i.e. spans both semesters).

Unresolved Issue: In `R27_Expanded_202510`, which is a Semester 1 dataset, the appearance of 2 and FY values raises an ambiguity:

- Does a 2 value within a Semester 1 dataset actually refer to a Semester 2 module that is pre-registered or cross-coded?
- Or is FY functionally equivalent to 2 in this context (i.e. both referring to modules delivered across the full academic year but reported under the same term)?

Action Required: This needs clarification from the source of the R27 extract (Registry or Student Records). The review should confirm:

1. Whether FY and 2 have distinct operational meanings in Banner or are treated equivalently.
2. Whether modules marked as FY in a Semester 1 extract (202510) should be included in Semester 2 reporting (202520) or counted only once.

3. If the current mapping (FY → Semester 1&2) should remain or be revised to align with official definitions.

Note: Until clarified, the cleaner maintains the current mapping but flags this as a point for data governance review.

Question 2: Duplicate records with same CRN but different Delivery Type (title embeds delivery marker)

Observation: In the cleaned R27 data, some records share the same CRN but differ only by **Delivery Type**. The **Title** field also embeds a delivery marker (e.g. “SEM 1” vs “SEM 2”), which makes otherwise identical rows appear different. Example:

| |
|------------------------------------|
| CMPU 1018 MAND SEM 1 Mathematics 1 |
| CMPU 1018 MAND SEM 2 Mathematics 1 |

Problem Statement: Are these distinct deliveries that should be retained separately, or are they artefacts of how delivery is encoded (once in **Delivery Type** and again inside **Title**) and therefore should be collapsed to a single row per CRN?

Clarifications Requested (from data owners / Registry):

1. Is **CRN** guaranteed to represent a unique delivery instance? If “yes”, then multiple rows with the same CRN should not differ in **Delivery Type**.
2. What is the authoritative source of delivery period: **Part of Term** or **Delivery Type**? Are both required, or is one canonical?
3. Should delivery markers embedded in **Title** (e.g. “SEM 1”, “SEM 2”, “MAND”) be treated as presentation-only and stripped/normalised, leaving delivery period to structured fields?
4. If **CRN** is unique and authoritative, should rows with the same CRN be de-duplicated by ignoring differences in **Delivery Type** and title-embedded markers?

Proposed Handling (pending decision):

- Treat **CRN** as the delivery-level key. For rows sharing the same CRN, collapse duplicates by:
 - Using mode (deterministic tie-break) for categorical fields (**Delivery Type**, **Campus**, **School Responsible**, etc.).
 - Normalising **Title** by removing embedded delivery markers (“SEM 1”, “SEM 2”, etc.) so that title reflects the module name only.
 - Asserting a single **STUDENT_COUNT** per CRN; if multiple counts exist, flag for review.
- If Registry confirms that **Delivery Type** differences alongside the same CRN are valid (e.g. split-mode delivery under one CRN), retain both records and document the reporting rule.

Risk if unresolved: Double-counting modules/students when **Title** encodes delivery and **Delivery Type** also differs for the same CRN.