# Assignment 2: Deduplication and Crawling

Prepared by
*Kushagra Gupta*
(2025909001)

Under the guidance of
*Prof. Anil Nelakanti*

Submitted in
partial fulfillment of the requirements
for the course of
**CS4.406: Information Retrieval and Extraction**

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

(November 2025)

# Activity 2.1: Deduplication

**Goal:** To find and group all records belonging to the same person from a noisy dataset of 5,000 entries.

**My Approach:** Followed a data-cleaning and linkage pipeline:

1. **Exploratory Data Analysis (EDA):** I first inspected the data and found many problems, including missing values, incorrect data types (like date_of_birth as a number), and typos in text fields (like 'nsw' vs. 'nws').
2. **Data Preprocessing:** I cleaned the entire dataset. I fixed all typos, converted dates and postcodes to standard string formats, and filled in missing values.
3. **Blocking:** To avoid comparing 12.5 million pairs, I used a "blocking" strategy. I grouped records by postcode, assuming matches would be in the same postal area.
4. **Pairwise Comparison:** Within these blocks, I used the recordlinkage library to compare pairs. I scored similarity on names, addresses, and date of birth using the Jaro-Winkler method.
5. **Decision & Clustering:** I set a rule that any pair with a total similarity score of 4.0 (out of 5.0) was a "match." I then used the networkx library to find all connected groups, or "clusters."

**Key Libraries Used:**

- pandas (for data loading and cleaning)
- recordlinkage & jellyfish (for blocking and comparing)
- networkx (for clustering the matched pairs)
- sklearn (for evaluation)

**Results & Insights:**

- **Insight:** The soc_sec_id column was the "answer key." The 5,000 records only had 2,291 unique soc_sec_ids. I used this to check my work.
- **Result 1 (Blocking**): My blocking strategy was very effective, reducing the number of comparisons by 99.87% (from 12.5 million to just 16,115).
- **Result 2 (Accuracy):** My algorithm found 1,059 clusters. When I compared this to the 2,291 "true" clusters, I got an Adjusted Rand Score (ARS) of 0.7307. This is a great result, showing my model's groups were very similar to the real ones. The lower cluster count means my model tended to "over-cluster," sometimes grouping two different people who just had very similar information.

# Activity 2.2: Crawling Summary (v1.2)

**Goal**

The assignment criteria changed from maximizing freshness to synchronizing visits with node updates. The goal was to minimize a custom metric: the Sum of Squared Lengths of contiguous update () and visit () events.

- Ideal Scenario: u, v, u, v (Streak lengths of 1). Metric = 1.0.

- Bad Scenario: u, u, u, v, v (Streaks of 3 and 2). Metric increases exponentially ($3^2 + 2^2$).

**Approach: The "Smart Predictive" Bot**

To minimize this metric, I could not simply crawl in a loop. I built a **Predictive Bot** using asyncio and aiohttp that models the behavior of each node.

1. **Data Gathering:** The bot parses the full HTML history of every node to extract timestamps of all previous updates.
2. **Modeling:** For each node, it calculates the **Average Update Interval** based on its history.
3. **Prediction:** It calculates a Predicted Next Update time ($T_{last} + Interval_{avg}$).
4. **Execution:** The bot sits in a loop. It only triggers a visit (refetch) if the current time has passed the predicted update time. This ensures we visit immediately after an update, breaking the u streak without creating a v streak.
5. **Safety Net:** If a node has insufficient history, the bot applies a heuristic (max wait time) to ensure data flow isn't completely stalled.

**Key Results**

The "Smart Predictive" strategy was highly effective compared to a naive looping strategy.

- **Metric Score:** The bot achieved an average custom metric score of **4.1862**.
    - This indicates that on average, the "streaks" of missed updates or redundant visits were very short (length ~ 2), proving the prediction logic was synchronized with the server's update frequency.
- **Behavior:** The logs showed the bot selectively visiting only 2-4 pages during quiet periods and ramping up activity only when update intervals were reached.
- **Conclusion:** The bot successfully modeled the hidden update parameters of the server nodes and minimized the error metric.

# Appendix

**GitHub Repo:** All the data, code file (ipynb and py), report.

https://github.com/SpyBeast07/HomeworkIRE/tree/main/deduplication_and_crawling