Implementation to string old ID’s with new ID’s:

Location: page 5. Exclude\_owners.py

**Third Tab:** “Annual Owner ID migration”

Dependencies: pip install rapidfuzz

Reads from: Excluded owners table

Writes to: Excluded owners table

\*\* no new tables\*\*

**Work Flow:**

Upload CSV —---> Load current EXCLUDE\_OWNERS (old IDs) │

Fuzzy match: OLD ID → New ID

Check for duplicate NEW\_OWNER\_IDs in CSV

Progress bar: “matching 3,928 ID’s…

↓

Review results:

✅ - High match: Auto approved

🟡 - Medium: review

🟠 - Low: review

❌ - No match: Remove from table (or flag for review)

**Fuzzy matching logic:**

old owner ID (Database) COMPARED TO New OWner ID(CSV)

├─ OWNER\_NAME ←───────→ ├─ OWNER\_NAME

├─ OWNER\_ADDRESS ←───────→ ├─ OWNER\_ADDRESS

├─ OWNER\_CITY ←───────→ ├─ OWNER\_CITY

└─ OWNER\_STATE ←───────→ └─ OWNER\_STATE

**Project overview and why RapidFuzz is best:**

The system prioritizes high-confidence matches “exact” before falling back to fuzzy techniques, minimizing false positives while maximizing recovery. We use RapidFuzz as the core library for its speed, ease of use, and tailored features like token\_sort\_ratio—ideal for handling reindex noise in names (Ex. reordered words, abbreviations like "3p Oil & Gas LLC" → "3P Oil Gas LLC").

* Speed for Scale: 10-100x faster than FuzzyWuzzy or built-in difflib due to C++ optimizations with pure Python bindings. Processes our dataset in seconds (vs. minutes), enabling quick iterations in VS Code without frustration.
* No Dependency
* Pure Python—no C compilation like FuzzyWuzzy's python-Levenshtein
* Cascading Fit: process.extractOne efficiently finds the "best match" per step

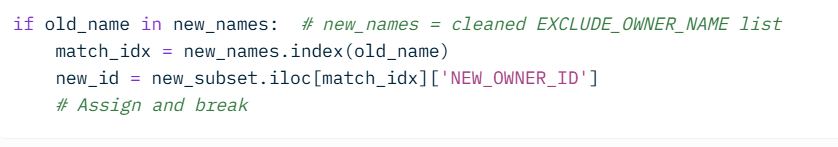
**Implementation steps**

**STEP ONE**

* Pip install rapidfuxx pandas openpyxl
* Data Prep:
  + Load missing report as old\_df (columns: Owner Name, Last Known Address, State, Previous Owner ID).
  + Load exclude comparison new\_df (dedupe on OLD\_OWNER\_ID for unique old→new mappings).
  + Clean all strings: Lowercase, strip "Attn:", "%", "-", punctuation (regex: r'[^\w\s]' → space), normalize spaces.
* Config Vars: Define NAME\_THRESHOLD = 80, ADDRESS\_THRESHOLD = 90 for tuning.

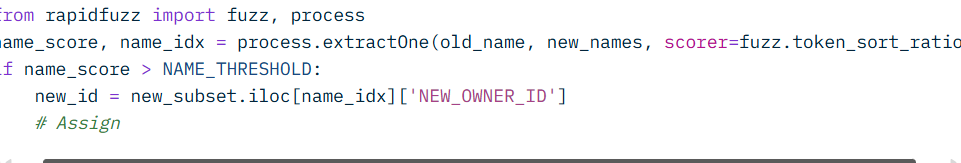
**STEP TWO: Direct Exact Match on OWNER\_NAME (High-Confidence Baseline)**

* For each missing owner, check if cleaned name exactly matches a cleaned EXCLUDE\_OWNER\_NAME in the state-blocked subset
* RapidFuzz role: Simple string equality. But prep with cleaning for consistency
* Output: If match, assign mapped\_new\_id = NEW\_OWNER\_ID, match\_step = 'EXACT\_NAME', name\_score = 100, status = ID\_STATUS (e.g., "ID\_CHANGED").
* Expected Yield: 10-20% of 2,484 missing
* Code Snippet:



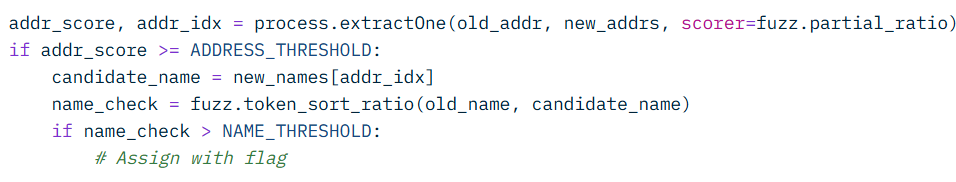
**STEP THREE: Fuzzy Match on OWNER\_NAME (>80% Similarity) – Cascade Fallback**

* Logic: If Step 1 fails, compute fuzzy similarity across state-blocked EXCLUDE\_OWNER\_NAMEs. Use token\_sort\_ratio to handle reindex noise (reorders, extras like "LLC" placement).
* process.extractOne(old\_name, new\_names, scorer=fuzz.token\_sort\_ratio) returns best match + score. Threshold >80% → assign new ID.
* Why Token Sort?: Sorts tokens before ratio (e.g., "Wells Fargo Attn: Trustee" → "attn fargo trustee wells" matches "wells fargo trustee" at 95%). Better than basic ratio (ignores order) for your data.
* Output: match\_step = 'FUZZY\_NAME', name\_score = score, status = ID\_STATUS
* Expected Yield: 20-40% additional recovery (e.g., "% Unknown; Attn: Operator" → "Unknown Operator Trust" at 85%).
* CODE SNIPPET:



**STEP FOUR: Address Match Rescue + Name Validation - Final Cascade**

* **LOGIC:** If Steps 1-2 fail, fuzzy match on full address (high threshold >=90% for "exact-ish" like PO Box variants). If hit, re-validate with name fuzzy (>80%) and flag for review.
* **RapidFuzz Role:** process.extractOne(old\_addr, new\_addrs, scorer=fuzz.partial\_ratio) for substrings (e.g., "PO Box 5190, San Antonio, TX" → "PO BOX 5190" at 92%). Then fallback to token\_sort on names.
* **Why Partial Ratio:** Matches subsets (ignores extra "Ste 110"), complementing token\_sort for addresses.
* **Output:** If valid, match\_step = 'ADDRESS\_REVIEW', address\_score = score, name\_score = check\_score, status = f"{ID\_STATUS} (REVIEW\_NEEDED)"
* **Expected Yield:** 10-20% more (address-stable cases like "9055 E Mineral Cir" despite name tweaks).

****

**STEP FIVE: Output and Validation**

* **Results DF:** Append all (matched/unmatched) to CSV with columns: Previous Owner ID, Owner Name, Last Known Address, State, mapped\_new\_id, match\_step, name\_score, address\_score, status.
* **Metrics:** Print value\_counts() on match\_step (e.g., UNMATCHED: 40%). Calculate recovery %: (matched / total\_missing) \* 100.
* **Review:** Export "ADDRESS\_REVIEW" rows to separate Excel for manual checks.
* **Turning Loop:** Rerun with threshold tweaks (e.g., NAME\_THRESHOLD=75 for 10% more fuzzy hits).

**\*\*** if <40% recovery, integrate Splink?