Read Me - Probate_RP_Prelim_Scoring

Here's a draft for a `README.md` file.

```
# Project Phoenix: Inherra Signal Engine - Phase 3 Record Linkage (Script 3)

## V1.0 Documentation

**Date:** May 28, 2025 (Adjust as needed)
**Lead Developer/Analyst:** [Your Name/Handle]
**AI Assistant (Gema):** Phase 3 PM & Scrum Master

## 1. Project Overview

This script (`Inherra Signal Engive V1.py` or `script3_v1.py`) is **Phase 3** of P
roject Phoenix. Its primary purpose is to perform advanced record linkage on
Real Property (RP) transaction data previously scraped by Script 2.

Given an input CSV containing RP records (where each row represents a part
y's involvement in a transaction, linked to an original probate lead), Script 3:
1.  Cleans and standardizes names.
2.  Generates phonetic keys (Soundex) for last names.
3.  Calculates various feature scores to determine the likelihood of a match be
tween a party on an RP record and the decedent from the original probate lea
d. Features include:
    *   Name similarity (last and first names)
    *   Date proximity (between RP filing date and probate filing date)
    *   Party role (bonus for Grantor)
    *   Instrument type relevance
    *   Search tier specificity (from Script 2's search)
```

4. Combines these features into a `match_score_total`.
5. Classifies each record into a `match_confidence_level` ('High', 'Medium', 'Low').
6. Outputs a comprehensive, enriched CSV file sorted by `match_score_total` descending, including all original data and the new scores/classifications.
7. Additionally outputs a smaller, stratified QA sample CSV for focused review.

The goal is to identify RP records that are highly likely to be associated with the probate decedent, primarily focusing on transactions where the decedent was a Grantor (as per Script 2's search strategy).

2. Script Functionality (V1.0)

Input:
* A semicolon-delimited (`;`) CSV file generated by Script 2. This file contains flattened RP records, where each row is a party to an RP transaction, enriched with data from the original probate lead.
* **Key expected input columns (among others):**
  * `probate_lead_decedent_first`, `probate_lead_decedent_last`, `probate_lead_filing_date`, `probate_lead_case_number`, `probate_lead_county`, `probate_lead_type_desc`, `probate_lead_subtype`, `probate_lead_status`, `probate_lead_signal_strength`
  * `rp_file_number`, `rp_file_date`, `rp_instrument_type`, `rp_party_type`, `rp_party_last_name`, `rp_party_first_name`
  * `rp_legal_description_text`, `rp_legal_lot`, `rp_legal_block`, `rp_legal_subdivision`, `rp_legal_abstract`, `rp_legal_survey`, `rp_legal_tract`, `rp_legal_sec`
  * `rp_signal_strength`, `rp_found_by_search_term`, `rp_search_tier`

Processing Steps:
1. **Dynamic File Handling:**
  * If no input CSV is specified via command-line, the script automatically searches for the most recently modified `.csv` file in the `DEFAULT_INPUT_FOLDER` (currently set to `"data/targeted_results/"` relative to script execution).
  * If no output CSV path is specified, a default name is generated based on the input filename and a timestamp (e.g., `input_filename_linked_YYYYMMDD_HHMMSS.csv`).

2. **Data Loading & Preprocessing:**
    * Loads the semicolon-delimited CSV into a pandas DataFrame.
    * Parses `probate_lead_filing_date` (expected format: `YYYY-MM-DD`) and `rp_file_date` (expected format: `MM/DD/YYYY`) into datetime objects.
3. **Name Cleaning (`clean_name_series`):**
    * Converts names to uppercase.
    * Removes common suffixes (JR, SR, II, III, IV) with optional periods.
    * Removes punctuation (retains letters, numbers, spaces, hyphens).
    * Normalizes multiple spaces to a single space and strips leading/trailing whitespace.
    * Creates `cleaned_...` versions of decedent and RP party names.
4. **Phonetic Key Generation (`generate_phonetic_keys`):**
    * Generates Soundex keys (using `jellyfish.soundex`) for cleaned last names (`soundex_decedent_last`, `soundex_rp_party_last`).
5. **Feature Score Calculation:**
    * **Name Similarity (`calculate_name_similarity_scores`):**
        * `name_last_score`: `rapidfuzz.fuzz.ratio` between cleaned decedent last name and cleaned RP party last name (0-100).
        * `name_first_score`: `rapidfuzz.fuzz.ratio` between cleaned decedent first name and cleaned RP party first name (0-100).
    * **Date Proximity (`calculate_date_proximity_score`):**
        * Calculates `days_apart` (absolute difference in days between `rp_file_date` and `probate_lead_filing_date`).
        * `date_proximity_score`: 100 if `days_apart < 180`, 50 if `days_apart < 365`, else 0.
    * **Party Role (`calculate_party_role_score`):**
        * `party_role_score`: 100 if `rp_party_type` is 'GRANTOR', else 0.
    * **Instrument Weight (`calculate_instrument_weight`):**
        * Assigns a score (0-100) based on `rp_instrument_type` using a predefined dictionary (e.g., "W/D" = 100, "DEED" = 95, "NOTICE" = 10, default = 25).
    * **Search Tier Weight (`calculate_search_tier_weight`):**
        * Assigns a score (0-100) based on `rp_search_tier` from Script 2 (e.g., "TIER_1" = 100, "TIER_2" = 75, "TIER_3" = 50, default = 25).
6. **Total Match Score (`calculate_match_score_total`):**
    * Calculates `match_score_total` as a weighted sum of the individual feature scores. Current weights:

    *   `name_last_score`: 0.25
    *   `name_first_score`: 0.10
    *   `date_proximity_score`: 0.20
    *   `party_role_score`: 0.10
    *   `instrument_weight`: 0.15
    *   `search_tier_weight`: 0.10
    *   Total score is clipped to be between 0 and 100 (currently max possible is 90 based on weights summing to 0.90).
7. **Confidence Classification (`classify_confidence_level`):**
    *   `match_confidence_level`: Categorizes `match_score_total` into 'High' (>=80), 'Medium' (>=60), or 'Low' (<60).
    *   `is_potential_decedent_match`: Boolean flag (TRUE if confidence is 'High' or 'Medium').
8. **Column Reordering & Output:**
    *   Reorders columns in the DataFrame for better readability, prioritizing scores, then legal info, then party/document/lead details.
    *   Saves the full, enriched, and sorted (by `match_score_total` descending) DataFrame to a semicolon-delimited CSV.
    *   Saves a separate, smaller (up to 25 rows) stratified QA sample CSV (`_QA_SAMPLE.csv`).

Output:
1. **Main Output CSV:** (e.g., `input_filename_linked_YYYYMMDD_HHMMSS.csv`)
    *   Contains all original columns from the input.
    *   Contains all new calculated columns (cleaned names, Soundex keys, individual feature scores, total score, confidence level, match flag).
    *   Rows are sorted by `match_score_total` in descending order.
    *   Semicolon-delimited.
2. **QA Sample CSV:** (e.g., `input_filename_linked_YYYYMMDD_HHMMSS_QA_SAMPLE.csv`)
    *   A subset of up to 25 rows from the main output, aiming for a stratified sample across High, Medium, and Low confidence levels.
    *   Semicolon-delimited.

3. Setup and Usage

Prerequisites:
* Python 3.x
* A virtual environment is highly recommended.

Installation:
1. Create and activate a virtual environment (e.g., `.venv_script3`):
   ```bash
   python3 -m venv .venv_script3
   source .venv_script3/bin/activate
   # For Windows: .venv_script3\\Scripts\\activate
   ```
2. Install required packages:
   ```bash
   pip install pandas rapidfuzz jellyfish numpy
   # python-Levenshtein is an optional C extension for rapidfuzz that improves speed.
   # If not already installed by rapidfuzz, you can add: pip install python-Levenshtein
   ```

Running the Script:
The script is designed to be run from the command line from its directory.

Option 1: Process the most recently modified CSV in the default input folder (`data/targeted_results/`) and generate a default output name:
```bash
python3 "Inherra Signal Engive V1.py"
```

(Ensure `DEFAULT_INPUT_FOLDER` in the script is correctly set relative to where you run the script, and the folder exists and contains Script 2 outputs.)

Option 2: Specify input and/or output files:

```
python3 "Inherra Signal Engive V1.py" --input path/to/your_input.csv --output path/to/your_output.csv
```

Or using positional arguments (if argparse is changed back or for older versions):

```
# python3 "Inherra Signal Engive V1.py" path/to/your_input.csv path/to/your_output.csv
```

Example using optional named arguments (current V1.0 setup):

- Process latest, default output name: `python3 "Inherra Signal Engive V1.py"`
- Process latest, specify output name: `python3 "Inherra Signal Engive V1.py" --output my_results.csv`
- Process specific input, default output name: `python3 "Inherra Signal Engive V1.py" --input data/targeted_results/specific_file.csv`
- Process specific input and output: `python3 "Inherra Signal Engive V1.py" --input specific_in.csv --output specific_out.csv`

VS Code `launch.json` Configuration:

To run/debug from VS Code easily (processing the latest input by default):
File:
`.vscode/launch.json`

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "name": "Python: Script3 Process Latest",
            "type": "debugpy", // Or "python" if using older debugger
            "request": "launch",
            "program": "${workspaceFolder}/Inherra Signal Engive V1.py", // Ensure this matches your script filename
            "python": "${workspaceFolder}/.venv_script3/bin/python",   // Explicit path to venv python
            "console": "integratedTerminal",
            "args": [], // Script will use internal defaults to find latest input & generate output name
            "cwd": "${workspaceFolder}"
        }
```

```
    ]
}
```

Remember to select the `.venv_script3` interpreter in VS Code for the workspace.

4. Key Functions

- `find_latest_csv_in_folder()`: Locates the newest CSV in the default input directory.
- `load_and_parse_dates()`: Loads data, ensures correct delimiter, parses date columns.
- `clean_name_series()`: Standardizes name strings.
- `generate_phonetic_keys()`: Creates Soundex keys.
- `calculate_name_similarity_scores()`: Computes fuzzy match scores for names.
- `calculate_date_proximity_score()`: Scores based on date differences.
- `calculate_party_role_score()`: Scores based on party type (e.g., Grantor).
- `calculate_instrument_weight()`: Scores based on RP document type.
- `calculate_search_tier_weight()`: Scores based on the Script 2 search tier.
- `calculate_match_score_total()`: Calculates the overall weighted score.
- `classify_confidence_level()`: Assigns High/Medium/Low confidence.
- `main()`: Orchestrates the entire process.

5. Future V1.1 Enhancements / Considerations

- Tune `WEIGHTS` and confidence thresholds based on QA review.
- Implement more robust name matching (e.g., `rapidfuzz.fuzz.token_set_ratio`, nickname dictionaries).
- Refine `date_proximity_score` logic based on domain insights.
- Add `legal_description_completeness` score.
- Normalize `match_score_total` if weights don't sum to 1.0 (if desired for a strict 0-100 scale).
- More sophisticated QA sampling.

- Error handling and logging improvements.

6. Next Major Phase (Script 4 / Phase 3 - Stage 2)

- **HCAD/Tax Roll Enrichment:** Take the output of this Script 3 (especially high-confidence links where the decedent was a grantor) and:
  1. Use the `rp_legal_description` to query HCAD (or similar).
  2. Retrieve current owner name and mailing address from HCAD.
  3. Compare HCAD owner to the `rp_grantee_name` from the deed.
  4. Output a list of properties with current HCAD owner and mailing address for outreach.

How to use this README:

1. Copy the text above.
2. Create a new file named `README.md` in your project root folder (`/Users/ayoodukale/Documents/Inherra/Python/Inherra scraper/`).
3. Paste the text into `README.md`.
4. **Customize it:**
   * Fill in `[Your Name/Handle]`.
   * Adjust the Date.
   * Double-check all script filenames, folder names, and especially the list of **Key expected input columns** to ensure they perfectly match your actual Script 2 output.
   * Review the `WEIGHTS` and scoring logic descriptions to ensure they reflect the final state of your V1.0 script.
5. Save the file.

This document will now serve as a great reference for your project!