match_my_statements

A smart CLI tool that automatically matches credit card statement entries with corresponding invoice files using AI-powered text similarity, date proximity, and amount comparison.

Overview

`match_my_statements.py` scans a credit card statement PDF (specifically German Hanseatic bank layout), looks for transaction lines, and matches each one against invoice files in a specified folder. The program uses:

- Text embedding similarity (OpenAI API)
- Date proximity comparison
- Amount matching with currency conversion
- Optional LLM-based verification for ambiguous matches

Installation

1. Clone this repository
2. Install required dependencies:

```
cd match_my_statements
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate
pip install -r requirements.txt
```

3. Create a `.env` file in the project directory with:

```
OPENAI_API_KEY=your_api_key_here
```

Usage

Basic usage:

```
python match_my_statements.py --statement
~/Downloads/booker/statements/April2023.pdf --invoices
~/Downloads/booker/creditcard
```

All options:

```
python match_my_statements.py --statement
~/Downloads/booker/statements/April2023.pdf \
                        --invoices ~/Downloads/booker/creditcard \
                        [--out
~/Downloads/booker/results/April2023_results.json] \
                        [--report
~/Downloads/booker/results/April2023_report.md] \
                        [--fx-cache ~/.fx_rates.json] \
                        [--threshold 0.6] \
                        [--dry-run] \
                        [--debug]
```

Parameters:

- `--statement`: Path to your Hanseatic Bank PDF statement
- `--invoices`: Directory containing renamed invoice files (usually ~/Downloads/booker/creditcard)
- `--out`: Custom path for JSON output (default: StatementName_results.json)
- `--report`: Custom path for markdown report (default: StatementName_report.md)
- `--fx-cache`: Path to store/load FX rate data (default: .fx_rates.json)
- `--threshold`: Matching confidence threshold (default: 0.6)
- `--dry-run`: Run without saving any output files
- `--debug`: Enable verbose debugging output during matching

Testing mode:

```
python match_my_statements.py --test
```

Input Requirements

Statement File

- PDF format
- German Hanseatic Bank credit card statement layout
- Contains transaction entries with dates, descriptions and amounts
- Typically placed in ~/Downloads/booker/statements/ folder

Invoice Files

The program expects invoice files named according to this pattern:

```
YY.MM.DD_method_amount.ccySlug.pdf
```

Example: `23.04.15_visa_9174_83.83EUR_shell_station.pdf`

Components:

- `YY.MM.DD`: Date in YY.MM.DD format
- `method`: Payment method (visa, mastercard, etc.)
- `amount.ccy`: Amount with currency code (e.g., 83.83EUR)
- `slug`: Description with underscores instead of spaces

These files are typically created by the companion `rename_my_invoices` tool and stored in the ~/Downloads/booker/creditcard/ or ~/Downloads/booker/giro/ folders.

Output Files

The program generates two output files in the same directory as the input statement:

1. JSON Results File (`StatementName_results.json`)

Contains the raw matching data:

- Full list of statement rows with metadata
- Full list of invoice files with metadata
- Map of statement row indices to matching invoice indices

2. Markdown Report (`StatementName_report.md`)

Human-readable summary with:

- Matching statistics overview
- Table of matched pairs
- Table of unmatched statement rows
- Table of unmatched invoice files

How It Works

1. **Statement Parsing**: Extracts transaction lines from PDF
2. **Invoice Loading**: Parses renamed invoice files from directory
3. **Text Normalization**: Normalizes merchant names (e.g., "Shell" → "fuel")
4. **Vector Embedding**: Creates semantic vector embeddings for each entry
5. **Matching**: Compares entries using:
    - Semantic similarity (cosine similarity of vectors)
    - Date proximity (closer dates score higher)
    - Amount similarity (with currency conversion when needed)
6. **Verification**: Uses GPT to verify ambiguous matches
7. **Report Generation**: Creates JSON data and markdown report

Example Workflow

1. Download credit card statement PDF to ~/Downloads/booker/statements/
2. Ensure invoices are properly renamed using `rename_my_invoices` and stored in ~/Downloads/booker/creditcard/
3. Run the matching process:

```
python match_my_statements.py --statement
~/Downloads/booker/statements/April2023.pdf --invoices
~/Downloads/booker/creditcard
```

4. Review the generated report (~/Downloads/booker/statements/April2023_report.md)
5. Check unmatched items and address any issues

Troubleshooting

- **Low match rate**:
  - Ensure invoice filenames follow the correct pattern
  - Try reducing the threshold with `--threshold 0.5`
  - Run with `--debug` to see matching details
- **Currency conversion issues**:
  - Check internet connection for FX rate API access
  - Use `--fx-cache` to specify a working cache file
- **PDF extraction issues**:
  - Ensure the statement follows Hanseatic Bank layout
  - Check the PDF is not password protected

Known Limitations

- Only works with Hanseatic Bank statement layout
- Requires OpenAI API access for embeddings and verification
- FX conversion uses daily rates, may not match exact bank rates
- Limited to transactions using 4-digit card suffix "9174"