Project: Introduction to Artificial Intelligence
Deadline: December 19, 2025

Description of the Real-World Problem

Manual data extraction from transport orders is time consuming, causes error, and requires repetitive work. Transport companies process hundreds of PDF documents weekly, each containing critical information like order numbers, dates, license plates, cities, and prices that must be manually entered into spreadsheets.

Goal:

Automate the extraction of structured data from transport order PDFs and automatically populate Google Sheets, reducing processing time from 2-3 minutes per document to seconds.

Motivation:

- Time savings: Manual processing of 100 PDFs takes 5 hours, automated system takes around 10 minutes
- Error reduction: Manual data entry has 5-10% error rate, automated system maintains consistency
- Cost efficiency: Reduces need for manual data entry labor

Input Data:

- Format: PDF documents
- Content: Transport orders containing:
    - Order numbers (format: XX/XXXXA)
    - Dates (DD.MM.YYYY)
    - License plates (Polish format: XXXXXXX)
    - Loading/unloading cities (Polish and German)
    - Freight prices (EUR)
- Languages: Polish and German

AI Domain:

- Primary: Information extraction, structured data extraction
- Secondary: Natural Language Processing (NLP), text extraction and Named Entity Recognition (NER)
- Task Type: Multi-label extraction + Entity recognition (cities, dates, numbers)

State of the Art

Approach 1: Large Language Model - Local Llama

During initial development, I experimented with running a local version of Llama for document analysis. The idea was interesting because LLMs can understand natural language context without predefined patterns. However, this approach proved impractical. Processing time was extremely long, taking approximately 5-6 minutes per document even with lighter versions of Llama. Moreover, the output quality was inconsistent and often incorrect. The model would hallucinate information, confuse similar numbers, or extract wrong city names. Additionally, running the model required significant computational resources (probably I had too low CPU), limiting deployment options. While the experiment showed LLMs could understand document structure, for our use case requiring speed and accuracy, this approach was rejected.

Approach 2: Pure Regex Pattern Matching

After leaving the LLM approach, I turned to traditional regular expressions. Regex patterns worked exceptionally well for structured fields - order numbers (XX/XXXXA), dates (DD.MM.YYYY), license plates (XX XXXXX), and prices achieved 98-99% accuracy because these formats are standardized. However, extracting city names using regex alone proved extremely challenging. Cities don't follow predictable patterns, vary in length, include special characters ("Frankfurt/Oder", "Kirchheim b. München"), and have diacritical marks ("Łódź", "München"). Attempts using postal codes sometimes captured street addresses instead, keyword based approaches extracted company names, and building comprehensive city lists was impractical. City extraction accuracy using pure regex was only 70-75%, requiring too many manual corrections. Additionally, regex patterns break when document formats vary slightly.

Approach 3: Pure spaCy NER (Named Entity Recognition)

I then explored using spaCy's Named Entity Recognition for all fields. SpaCy uses pre-trained machine learning models (`pl_core_news_sm`, `de_core_news_sm`) that identify entities like locations, dates, and numbers. SpaCy performed well for cities, achieving around 85-88% accuracy and handling variations that regex struggled with. However, applying spaCy to all fields caused problems. Processing time increased to 4/5s per document versus milliseconds for regex. More importantly, spaCy struggled with structured fields, order numbers would be split into tokens, dates weren't preserved in exact format, and price accuracy was lower than regex. The probabilistic nature also caused inconsistent results, problematic for production use requiring reproducibility.

Description of the Chosen Concept

Hybrid Approach (Regex + spaCy NER)

The final system benefits the strengths of both methods. Regex handles all structured fields (order numbers, dates, plates, prices) achieving around 98-99% accuracy. For cities, a two-tier approach first attempts regex with postal codes, then uses spaCy NER when needed. This achieves around 94% overall accuracy with fast processing (3-4 seconds per document). The system is maintainable, regex handles static extraction reliably while spaCy provides changing fallback for challenging city extraction. Required data (pdfs of transport orders) are accessible to us thanks to my father's transport company which gives us a large dataset with different variants of test data. Each PDF produces JSON with six fields: order number, date, license plate, loading city, unloading city, and price. Data exports to Google Sheets organized by license plate in weekly tables with chronological sorting or displayed in the console output.

General algorithm:

PDF Document

↓

[PyPDF2] Text Extraction

↓

[Regex Extractor] → Order Number, Date, License Plate, Price

↓

[spaCy NER] → Loading City, Unloading City

↓

[Data Processor] → Group by License Plate

↓

[Google Sheets API] → Export to Spreadsheet

This concept is already implemented in transport company and saves plenty of time. As a testing procedure I've tried a trial-and-error method and if some mistake occurred I changed the pattern of the regex or improved the spacy method. Possible limitations may occur when the system encounters significantly different types of transport orders than those used during development and testing. For instance, if a new client or logistics platform uses a completely different document template, the location of key information such as loading and unloading places may appear in different sections or under differently named headers. However, I believe these adaptations would not be particularly difficult to implement.