SEO Proofreader Prototype Report

By Merve EMRE

1. Introduction: Understanding the Challenge

Trustoo's content strategy relies on high-quality "Cost Pages" and "City Pages" that require a time-consuming manual SEO proofreading process. This case study presents a prototype for an automated SEO Proofreader, designed to analyze articles against official checklists, reduce manual effort, and accelerate the content pipeline by providing instant, actionable feedback.

2. My Approach and Technical Choices

My approach was to build a robust, lightweight, and easily reproducible prototype using a single Jupyter Notebook. This choice prioritizes clarity and ease of evaluation, allowing anyone to run the entire analysis and see the code, logic, and results in one place.

The core of the prototype was developed using Python 3, leveraging a minimal set of powerful, open-source libraries:

- Pandas: Chosen for its simplicity and robustness in handling structured data. It was used to read the keyword sets from .csv files into a clean, usable list format. Although the case provided Excel files, I chose to work with .csv versions to minimize dependencies.
- Re (Regular Expressions): This standard Python library was the primary engine for most text analysis. It is incredibly efficient for pattern matching and was used to find and validate specific formats (e.g., page titles, meta descriptions, pricing, and percentages) without the overhead of larger NLP libraries.
- OS and Collections: Standard libraries used for file path management (os.path.join) to ensure the code runs on any operating system, and for counting word frequencies (Counter) to detect keyword repetition.

I deliberately avoided complex NLP libraries for this prototype. The goal was to demonstrate that the majority of the checklist rules are deterministic and pattern-based, and can be effectively automated with simpler, faster tools. This approach also keeps the prototype lightweight and free from heavy dependencies.

Data Handling: The provided sample articles were treated as raw .txt files and the keyword sets as .csv files. This decision was made to ensure the prototype works with the most basic, universal file formats, avoiding the need for specialized libraries to parse .docx or .xlsx files, thus simplifying the setup for anyone running the code.

3. Mapping Checklist Items to Code

The core of the prototype is a series of functions, each mapping to a specific category in the SEO checklist.

- Page Title and Meta Description (check_page_title, check_meta_description): These functions use regex to find lines prefixed with title: and meta:. They then check for length, presence of the main keyword, use of separators (|), and word repetition.
- Headings and Keywords (check_headings_and_keywords): H1 Check; The H1 is assumed to be the first non-meta/title line of the text. The code performs a case-insensitive, exact-match comparison against the main keyword (the first keyword in the provided list). Keyword Density; regex is used to count the occurrences of the main keyword and calculate its density relative to the total word count. This helps identify both keyword stuffing (>2.5%) and under-utilization (<0.5%).
- Content Quality & Links (check_content_quality_and_links): Internal Linking; A regex search looks for URLs containing "trustoo" or simple text mentions of "top 10" to verify internal links. CTAs and Readability; The code scans for a predefined list of Call-to-Action phrases and identifies overly long paragraphs (based on a word count threshold) to encourage scannability.
- Formatting Rules (check_formatting_rules): Price and Percentage; regex patterns were designed to validate the specific formatting of prices (e.g., € 1.200,-) and percentages (e.g., 30%), flagging common errors like incorrect spacing.

4. Limitations and Next Steps

Limitations:

- This prototype successfully demonstrates the concept but has clear limitations that provide a roadmap for future development.
- Simplistic Page Type Detection: Page type is currently determined by the presence of 'cost' or 'city' keywords, which can lead to misclassification, as seen in the test reports.
- Input Format Dependency: The tool requires title: and meta: prefixes in the input files to analyze those elements. The low scores for the city pages are a direct result of this data being absent.
- Heuristic-Based Analysis: Heading detection is based on simple heuristics (e.g., line length) and cannot differentiate between H2, H3, etc. This is a clear area for improvement.
- Incomplete Checklist Coverage: Subjective and complex checks, such as grammar/spelling, assessing content "value" against competitors, and advanced formatting rules (e.g., writing out numbers), were intentionally omitted from this prototype.

Next Steps:

- Robust Heading Parsing: Integrate a library like BeautifulSoup (if input is HTML) or python-docx to analyze actual document structure and heading tags (H1, H2, H3) for more accurate analysis.
- Integrating NLP for Deeper Analysis: Incorporate a library like language-tool-python for grammar and spelling checks. A more advanced step would be to use topic modeling to verify that all subtopics from the keyword research are covered.
- Developing a User-Friendly Interface: Build a simple web application using Flask, allowing editors to paste text or upload files and receive an interactive report without running any code.
- Refine Scoring Algorithm: Collaborate with the SEO team to fine-tune the weights of each checklist item in the overall_score calculation, ensuring it accurately reflects business priorities.