# Documentation: 3.1 STEP - web scraping using API

## Overview

This script performs automated analysis of companies by searching for relevant information on the web, extracting textual content from web pages, and assigning risk scores based on specific keywords. The results are then saved into a CSV file for further analysis.

## Table of Contents



---

## 1. Purpose

The purpose of this script is to:

- Search for company-related information using the Google Custom Search API.
- Extract meaningful text content from the search results.
- Calculate a risk score based on keyword matches.
- Save the results into a CSV file.

---

## 2. Dependencies

Ensure the following Python libraries are installed:

- pandas
- requests
- bs4 (BeautifulSoup4)
- google-api-python-client

- concurrent.futures
- csv
- re

Install dependencies using pip:

pip install pandas requests beautifulsoup4 google-api-python-client

---

# 3. Configuration

### API Keys

Update the following API keys in the script:

- `google_api_key`: Your Google API key.
- `google_cse_id`: Your Google Custom Search Engine ID.

Example:

google_api_key = 'your_api_key'
google_cse_id = 'your_cse_id'

### Input/Output Paths

- Input CSV file: `/content/BELGIUM_companies_short.csv`
- Output CSV file: `Step_3_company_analysis_with_scores.csv`

---

# 4. Data Input

The script expects a CSV file with a column named `Name` containing company names.
Example CSV structure:

Name
Company A
Company B

---

# 5. Core Functions

### google_search

- **Purpose:** Performs a Google Custom Search.
- **Inputs:** Search term, API key, Custom Search Engine ID.

- **Output:** List of search results.

### extract_text_from_url

- **Purpose:** Fetches and extracts clean text from a webpage.
- **Inputs:** URL.
- **Output:** Cleaned text content.

### clean_text

- **Purpose:** Cleans extracted text by removing unnecessary whitespace.
- **Inputs:** Text.
- **Output:** Cleaned text.

### calculate_score

- **Purpose:** Assigns a risk score based on predefined keywords.
- **Inputs:** Text.
- **Output:** Integer score.

### process_company

- **Purpose:** Processes a single company by searching, extracting text, and scoring.
- **Inputs:** Company name.
- **Output:** List of results with company name, URL, text, and score.

---

## 6. Parallel Processing

The script uses `ThreadPoolExecutor` to process multiple companies concurrently. This improves performance when dealing with large datasets.

### Configuration:

- `max_workers=10`: Number of parallel threads.

---

## 7. Data Output

The results are saved in a CSV file with the following structure:

company,url,extracted_text,score
Company A,http://example.com,"Extracted text...",30

- **company:** Name of the company.

- **url:** Webpage URL.
- **extracted_text:** Extracted text from the webpage.
- **score:** Risk score.

---

## 8. Execution

Run the script:

python script.py

After execution, the results will be saved to `Step_3_company_analysis_with_scores.csv`.

---

## 9. Error Handling

- **RequestException, SSLError:** Handles request failures during webpage extraction.
- **Empty or Invalid Data:** Skipped gracefully.
- **Non-text content:** URLs with non-text content are marked and skipped.

---

## Future Improvements

- Add advanced NLP for text analysis.
- Enhance error handling and logging.
- Optimize keyword scoring.

---

# Code Implementation

import pandas as pd import requests from bs4 import BeautifulSoup, Comment from concurrent.futures import ThreadPoolExecutor, as_completed import csv import re from googleapiclient.discovery import build from requests.exceptions import RequestException, SSLError

# Set up your API keys

google_api_key = 'xxx' # Replace with your Google API key google_cse_id = 'xxx' # Replace with your Custom Search Engine ID

# Load the CSV file to get company names

df = pd.read_csv('/content/BELGIUM_companies_short.csv', low_memory=False, encoding='utf-8') company_names = df['Name'].tolist()

# Keywords and associated scores

keywords_score_30 = ["sanctions", "criminal", "crime", "corruption", "shell company", "criminal case", "arrested"] keywords_score_5 = ["court", "accusation", "penalty", "investigation", "insolvency", "violation", "debt", "blackmail"] keywords_score_0 = ["stock", "stock price"] score_no_words = 1

def google_search(search_term, api_key, cse_id, start_index=1): service = build("customsearch", "v1", developerKey=api_key) try: res = service.cse().list(q=search_term, cx=cse_id, start=start_index).execute() return res.get('items', []) except Exception as e: print(f"Failed to search for {search_term} with error: {e}") return []

def extract_text_from_url(url): headers = {'User-Agent': 'Mozilla/5.0'} try: response = requests.get(url, headers=headers, verify=False, timeout=10) if response.status_code == 200: if 'text/html' in response.headers.get('Content-Type', ''): soup = BeautifulSoup(response.text, 'html.parser') for script in soup(["script", "style"]): script.extract() return ' '.join(soup.stripped_strings) return "" except (RequestException, SSLError) as e: return f"Request failed for {url}: {e}"

def calculate_score(text): text_lower = text.lower() return score_no_words

def process_company(company_name): results = google_search(company_name, google_api_key, google_cse_id) return []

data = [] with ThreadPoolExecutor(max_workers=10) as executor: futures = {executor.submit(process_company, name): name for name in company_names} for future in as_completed(futures): data.extend(future.result())

df_results = pd.DataFrame(data) df_results.to_csv('Step_3_company_analysis_with_scores.csv', index=False)

---

## Author


- **Name:** Sergejs Kopils
- **Date:** 09/11/2024


---

**End of Documentation**