

LOGO SIMILARITY

POPA ALEXIA LARISA

THE PURPOSE OF THIS PRESENTATION IS NOT TO BORE YOU; THEREFORE, I WILL AIM
TO HIGHLIGHT THE KEY IDEAS AND PROVIDE DETAILED EXPLANATIONS WHERE
NECESSARY :))



OVERVIEW

The following slides will illustrate the necessary steps for solving this challenge, as well as the chosen approach and why.

Additionally, I will explain the functioning of the algorithm, with the aim of highlighting the technical details.

STEP 1

THE DECRYPTION OF THE FILE

The algorithm processes a .parquet file containing domain or URL data. It loads the file into a Pandas DataFrame and extracts the column named 'domain'. Each entry is cleaned by removing prefixes like www. and https://, stripping query parameters and subpaths, and converting to lowercase.

After cleaning, it filters out invalid entries, removes duplicates, and sorts the results. Finally, the cleaned list of unique domains is saved to a text file (cleaned_websites.txt). The algorithm also includes error handling for missing dependencies and incorrect column names.

The algorithm was implemented in Python due to its rich ecosystem of data processing libraries (such as pandas and re) and its readability, which makes it well-suited for rapid development and data manipulation tasks. Additionally, I prefer this method because I believe it is significantly easier for me, which enables me to perform the task more effectively.

```
1 1 import pandas as pd
2 2 import re
3 3
4 4 input_parquet_file = "logos.snappy.parquet"
5 5 output_txt_file = "cleaned_websites.txt"
6 6
7 7 try:
8 8     df = pd.read_parquet(input_parquet_file)
9 9
10 10 url_column_name = 'domain'
11 11 if url_column_name not in df.columns:
12 12     print(f"Error: Column '{url_column_name}' not found in Parquet file.")
13 13     print("Available columns are:", df.columns.tolist())
14 14     exit()
15 15
16 16 raw_urls = df[url_column_name].tolist()
17 17
18 18 def clean_url(url):
19 19     if isinstance(url, str):
20 20         url = url.strip()
21 21         url = re.sub(r'^www\.', '', url, flags=re.IGNORECASE)
22 22         url = re.sub(r'^(https?://)', '', url, flags=re.IGNORECASE)
23 23         url = url.split('/')[0]
24 24         url = url.split('?')[0]
25 25         url = url.strip().lower()
26 26         return url
27 27     return None
28 28
29 29 cleaned_urls = [clean_url(url) for url in raw_urls if clean_url(url) is not None]
30 30
31 31 unique_cleaned_urls = sorted(list(set(cleaned_urls)))
32 32
33 33 with open(output_txt_file, "w", encoding="utf-8") as f:
34 34     for url in unique_cleaned_urls:
35 35         f.write(url + "\n")
36 36
37 37     print(f"Successfully cleaned and saved {len(unique_cleaned_urls)} unique URLs to {output_txt_file}")
38 38
39 39 except Exception as e:
40 40     print(f"An error occurred: {e}")
41 41     print("Please ensure 'pandas' and 'pyarrow' are installed (`pip install pandas pyarrow`)")
```

STEP 2

EXTRACT LOGOS

In the initial iteration of the logo extraction script, the success rate for logo retrieval was approximately 53%. Although the script processed all specified URLs, it successfully extracted 1,816 logos, with errors reported for the remainder. This observation underscored the necessity for enhancing the script's robustness.

Analysis of the error log revealed a significant prevalence of connection-related failures. Furthermore, it was hypothesized that a considerable number of websites might not store their logos in standard or easily discoverable locations within their HTML structure.

--- Summary ---

Total URLs processed: 3416

Logos downloaded: 1816

Logos skipped (already existed): 0

Errors / No logo found: 1600

Check 'download_log.txt' for successful downloads.

Check 'download_errors.txt' for errors.

PS C:\Users\A\OneDrive\Desktop\logo project> □

0 ▲ 0

```
ERROR: aad-insurance.com - No logo URL found
ERROR: http://aamcoglendaleca.com - HTTP 403
ERROR: http://aamcogreensburgpa.com - Timeout
ERROR: aamcoflascruces.com - No logo URL found
ERROR: http://aamcotracyca.com - HTTP 403
ERROR: abc-autoglas.de - No logo URL found
ERROR: http://acelltherapy.com.au - HTTP 520
ERROR: http://acetoolonline.com - HTTP 403
ERROR: http://acuraofpeabody.com - HTTP 405
ERROR: http://acuraoframsey.com - HTTP 403
ERROR: http://acuraofwappingersfalls.com - HTTP 403
ERROR: http://acuraofwestchester.com - HTTP 403
ERROR: adecco.com.vn - No logo URL found
ERROR: http://adra.org.au - HTTP 403
ERROR: adra.org.pt - Failed to download logo from https://adra.org.pt/wp-content/uploads/2021/11/cropped-WhatsApp-Image-2021-11-01-23.jpeg-32x32.png
ERROR: aeccglobal.co.th - Failed to download logo from http://aeccglobal.co.th/images/fav-icon.png
ERROR: affidea.ch - No logo URL found
ERROR: affidea.es - No logo URL found
ERROR: affidea.gr - No logo URL found
ERROR: affidea.lt - No logo URL found
ERROR: http://africell.ao - Timeout
ERROR: http://aireserv.ca - HTTP 403
ERROR: ajaxtocco.de - No logo URL found
ERROR: http://alcirclebiz.com - Timeout
```

STEP 2

EXTRACT LOGOS

In order to increase the results, I managed to build an algorithm that

1. Reads URLs from an input file.
2. Normalizes each URL (adds http:// if missing).
3. For each URL:
 - Sends a HEAD request (fast check).
 - If the response is not clearly OK (e.g. 403, 301), tries a full GET request.
4. Uses multiple threads (e.g. 50) to speed up the process.
5. Prints the status of each URL (OK or error).
6. Saves all functional URLs to a new file.
7. Reports success rate and time taken.

```
[FINAL] Funcționale: 2595 / 3416 (75.97%)
[TEMP] Total: 102.09 secunde
[ATENȚIE] Procent sub 97%! Verifică sursa URL-urilor.
[SALVAT] URL-uri funcționale scrise în: urls_funcționale.txt
PS C:\Users\A\OneDrive\Desktop\logo project>
```

UNFORTUNATELY, THIS APPROACH DID NOT YIELD A SIGNIFICANT SUCCESS RATE IN THE FINAL OUTCOME, WHICH IS WHY I CHOSE TO DISCONTINUE ITS USE.

DURING THE DEVELOPMENT OF THE LOGO EXTRACTION ALGORITHM, SEVERAL KEY ISSUES WERE ENCOUNTERED:

- MANY WEBSITES FAILED TO LOAD OR BLOCKED AUTOMATED ACCESS, LEADING TO A HIGH NUMBER OF REQUEST TIMEOUTS.
- LOGO DETECTION WAS OFTEN UNSUCCESSFUL DUE TO DYNAMIC CONTENT, NON-STANDARD HTML STRUCTURES, OR HIDDEN IMAGES.
- THE SCRAPING PROCESS WAS SLOW AND RESOURCE-INTENSIVE, ESPECIALLY WHEN SCALING TO THOUSANDS OF URLs.
- CLUSTERING LOGOS BASED ON VISUAL SIMILARITY PROVED DIFFICULT DUE TO INCONSISTENCIES IN IMAGE SIZE, FORMAT, AND

QUALITY.

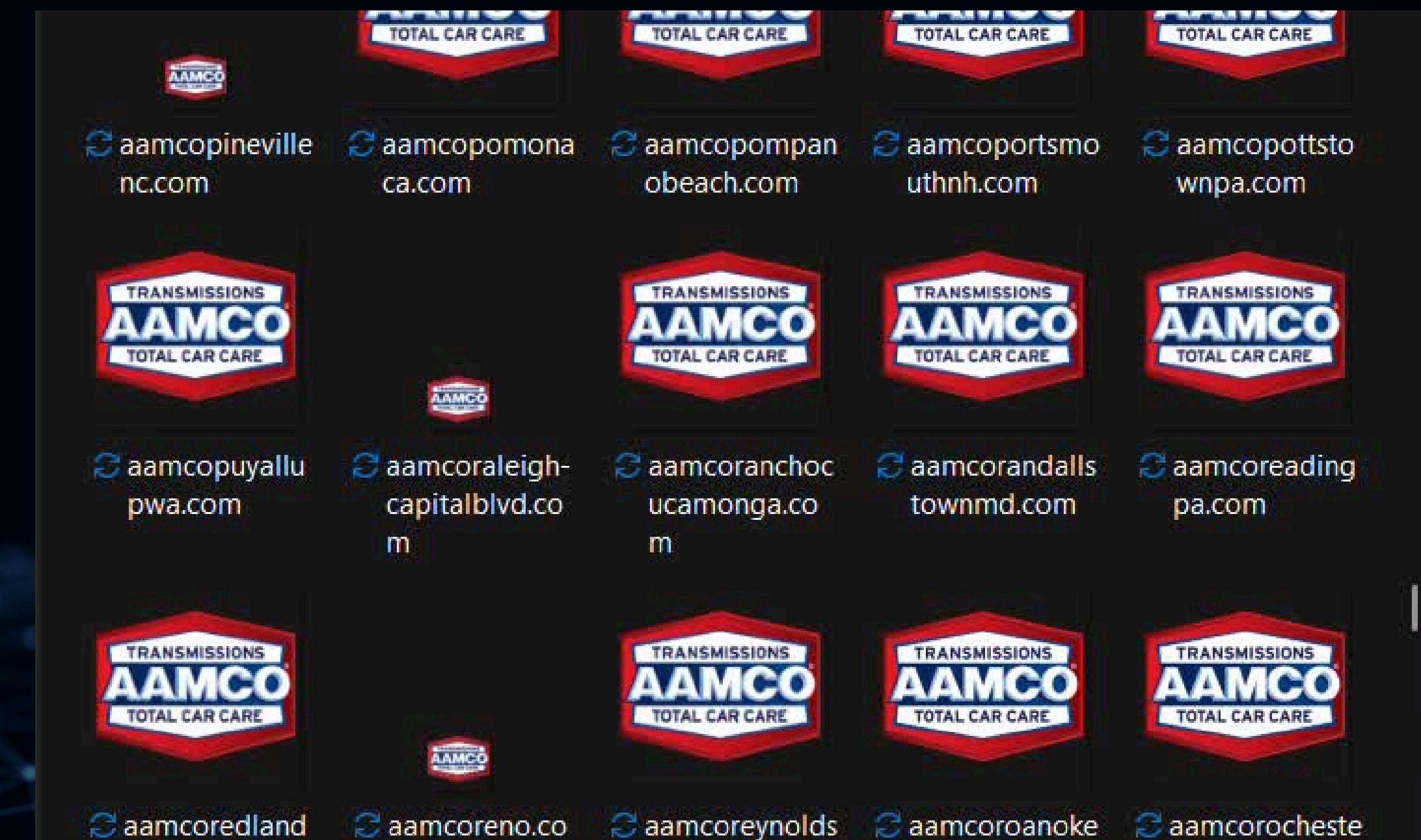
THESE CHALLENGES SIGNIFICANTLY REDUCED THE ALGORITHM'S EFFECTIVENESS AND REQUIRED RE-EVALUATION OF THE CHOSEN APPROACH.

IN THE LIGHT OF THE ABOVE, I CONSIDERED THE BEST OPTION TO CHOOSE THE VERSION WITH 53% RESULTS AND MOVE FORWARD WITH THE NEXT STEP: CLUSTERING

STEP 3

CLUSTERING

- Feature Extraction (Non-ML): It uses `imagehash.phash` to calculate a perceptual hash for each logo. This hash is a unique "fingerprint" that represents the image's visual content. Visually similar images will have similar hashes.
- Grouping (Non-ML Heuristic): It iterates through all logos. For each logo, it compares its hash to the "representative hash" of existing groups. If the Hamming distance (number of differing bits) between the current logo's hash and a group's representative hash is below a configurable `MAX_HASH_DISTANCE` threshold, the logo is added to that group. Otherwise, a new group is created.
- Output: It creates separate subdirectories for each identified group (cluster) and copies the respective logos into them. The `MAX_HASH_DISTANCE` parameter can be adjusted to control how strictly logos are grouped based on visual similarity.



THANK YOU!



I appreciate your time and consideration for this position. Please do not hesitate to contact me should you require any additional information. This challenge provided a valuable learning experience.