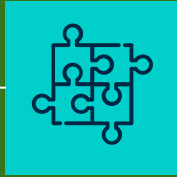


# SEARCH ENGINE OPTIMIZATION

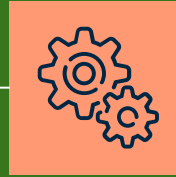
Ayush Majumdar, Akshaj Joshi, Anunay  
Akhaury, Corey Chen, Tyler Guo, Ankita  
Khatri, Aimee Tsai, Sarayu Mummidi

Why was the SEO expert bad at telling jokes? Because the punchline was always buried on page 2!



01

PROJECT  
INTRODUCTION



02

LDA FOCUS  
AND PROJ



03

PROJECT  
DEMONSTRATION

# Why did the SEO expert go broke! He kept buying the best keywords but couldn't convert his leads!



Search Engine Optimization (SEO) is the practice of enhancing a website or web content to improve its visibility and ranking in search engine results pages (SERPs). The goal of SEO is to attract more organic (non-paid) traffic from search engines by ensuring that the content appears prominently when relevant keywords or phrases are searched.

# TOOLS WE USED

**matplotlib**  
data visualization



**Python - LDA**  
pandas, diffbot, google  
client-api, numPy, sklearn,  
aiohttp, asyncio

**MongoDB**  
database management



**Django**  
web framework



# OUR PROCESS PART 1

- Extracted the HTML and code from any submitted URL and get information from the HTML
  - Created a broad field of the industry of the URL
- For example, if the input is safeway.com, the output could be “grocery retail sector for food and housing products”.
- Inputted “grocery retail sector for food and housing products” into Google based on output
  - Scrapes for valuable metadata and potential keywords from the first X amount of results in the search engine
  - Outputs wide high-click level keywords and information.

# Web-Scraping – How do web scrapers stay calm under distress?

## They know how to handle a lot of requests!

- To extract industry information from websites using our web scraper, we employed a combination of powerful tools and techniques. Firstly, we used BeautifulSoup, a Python library for parsing HTML and XML documents, to navigate and extract data from the website's HTML structure. This allowed us to locate and retrieve the specific sections containing industry-related information. Additionally, we integrated Selenium, a web testing framework, to handle dynamic content and automate browser interactions, enabling us to access industry details that require user interaction or are rendered dynamically with JavaScript. To enhance data accuracy, we implemented regular expressions (regex) for precise pattern matching and text extraction, ensuring that we accurately captured the relevant industry data. Finally, we utilized Pandas, a data manipulation library, to clean and organize the extracted information into a structured format, facilitating further analysis and utilization in our projects.

```
# Async function to get industry from Diffbot
async def get_industry_from_diffbot(session, url, api_key, retries=3, backoff_factor=0.3):
    api_url = "https://api.diffbot.com/v2/article"
    params = {'token': api_key, 'url': url}
    for attempt in range(retries):
        try:
            async with session.get(api_url, params=params) as response:
                response.raise_for_status()
                data = await response.json()
                return data['objects'][0] if 'objects' in data and data['objects'] else None
        except (aiohttp.ClientError, aiohttp.ServerDisconnectedError) as e:
            await asyncio.sleep(backoff_factor * (2 ** attempt)) # Exponential backoff
            raise aiohttp.ServerDisconnectedError(f"Failed to connect to server after {retries} attempts")

# Function to extract specific industry
def get_specific_industry(data):
    if data:
        tags_list = data.get('tags', [])
        return ', '.join([str(tag.get('name', '')) for tag in tags_list]) if tags_list else None
    return None
```

```
# Async function to crawl website and get industries
async def crawl_website(start_url, api_key):
    async with aiohttp.ClientSession() as session:
        industries = set()

        while len(industries) < 3: # Loop until there are at least three distinct industries
            article_data = await get_industry_from_diffbot(session, start_url, api_key)

            if article_data:
                general_industries = article_data.get('categories', [])
                specific_industry = get_specific_industry(article_data)

                if specific_industry:
                    industries.update(specific_industry.split(', '))
                elif general_industries:
                    industries.update([category.get('name', '') for category in general_industries])

        return list(industries)[:3] # Return the first three unique industries
```

# Web-Scraping Using Google-Client

**How do Google Clients communicate with their search engines? With great queries!**

- We used a Google-Client API to plug in our specified industry from our industry search tool into a google search engine, in order to extract links to further web scrape.
- To make coding against these APIs easier, Google provides client libraries that can reduce the amount of code you need to write and make your code more robust.

```
# Function to perform Google Custom Search
def google_search(search_terms, api_key, cse_id):
    service = build("customsearch", "v1", developerKey=api_key)
    combined_search_query = ' OR '.join(search_terms)
    res = service.cse().list(q=combined_search_query, cx=cse_id).execute()
    return [item['link'] for item in res['items'][:5]]
```

# Implementing Latent Dirichlet Allocation (LDA) Model

- LDA is a generative statistical model that allows sets of observations to be explained by unobserved groups that explain why some parts of the data are similar, it is used to discover the underlying topics in a collection of documents
- Preprocessed and tokenized the text data from our database. Then created a document-term matrix. Trained an LDA model on this matrix to identify topics and their associated keywords.
- Analyze the relevance and frequency of these keywords, then recommend them based on the users inputted website and cross reference to suggest the best keywords.



## LDA PART 2 – What do you call an LDA model that loves to gossip? A topic generator!

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from gensim import corpora
from gensim.models.ldamodel import LdaModel

# Sample documents
documents = [
    "Human machine interface for lab abc computer applications",
    "A survey of user opinion of computer system response time",
    "The EPS user interface management system",
    "System and human system engineering testing of EPS",
    "Relation of user perceived response time to error measurement",
    "The generation of random binary unordered trees",
    "The intersection graph of paths in trees",
    "Graph minors IV Widths of trees and well quasi ordering",
    "Graph minors A survey",
]

# Download NLTK stopwords
nltk.download('punkt')
nltk.download('stopwords')

# Preprocess documents
stop_words = set(stopwords.words('english'))

def preprocess(document):
    tokens = word_tokenize(document.lower())
    filtered_tokens = [token for token in tokens if token.isalnum() and token not in stop_words]
    return filtered_tokens

processed_docs = [preprocess(doc) for doc in documents]

# Create a dictionary representation of the documents
dictionary = corpora.Dictionary(processed_docs)

# Create a bag-of-words representation of the documents
bow_corpus = [dictionary.doc2bow(doc) for doc in processed_docs]
```

### OUTPUT:

#### Topic: 0

Words: 0.080"user" + 0.080"system" + 0.078"graph" + 0.078"response" + 0.078"time" + 0.041"opinion" + 0.041"computer" + 0.041"interface" + 0.041"survey" + 0.041"minor"

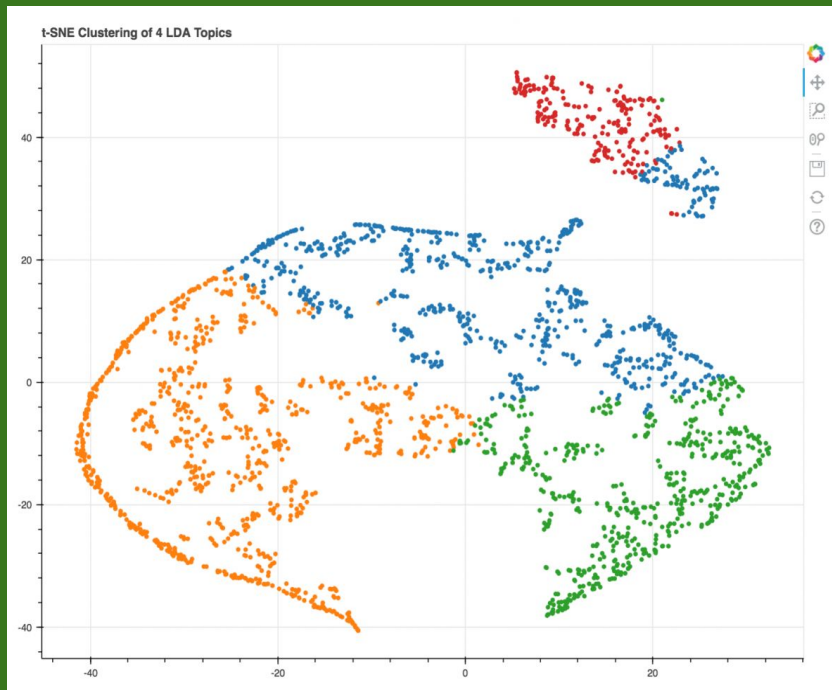
#### Topic: 1

Words: 0.110"system" + 0.090"human" + 0.090"eps" + 0.050"machine" + 0.050"interface" + 0.050"applications" + 0.050"lab" + 0.050"abc" + 0.050"computer" + 0.050"engineering"

#### Topic: 2

Words: 0.100"graph" + 0.100"trees" + 0.070"minors" + 0.070"survey" + 0.040"generation" + 0.040"intersection" + 0.040"binary" + 0.040"random" + 0.040"unordered" + 0.040"paths"

# LDA PART 3



The blue cluster at the top is distinctly separated from the other clusters, suggesting that the documents in this cluster are quite different from those in the other clusters.

The orange cluster on the left and the green cluster on the right are relatively well-separated, indicating clear topic distinctions.

There is some overlap between the red cluster and the green cluster, which could suggest that some documents share characteristics of both topics or that the topics are somewhat related.

# LDA PART 4

	Keyword 1	Keyword 2	Keyword 3	Keyword 4	Keyword 5	Keyword 6	Keyword 7	Keyword 8	Keyword 9	Keyword 10
	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----
Topic 1	applications	blockchain	management	chain	supply	machine	cybersecurity	algorithms	learning	care
Topic 2	healthcare	artificial	intelligence	patient	revolutionizing	innovations	financial	transforming	banking	sector
Topic 3	agriculture	breakthroughs	biotechnology	transportation	future	electric	vehicles	retail	shaping	trends
Topic 4	data	marketing	strategies	computing	cloud	enterprise	solutions	advancements	renewable	energy

This table shows the top 10 keywords for each of the 4 topics identified by the LDA model.

# What is an SEO expert favorite fruit? Low-hanging keywords!

- We use `aiohttp.ClientSession` to perform asynchronous HTTP requests to fetch data from articles using provided links and an API key. This approach allows multiple requests to be handled concurrently, improving efficiency when dealing with a list of URLs.
- - After retrieving articles, we parsed the data to extract and accumulate keywords from the 'tags' section of each article. The process stops collecting keywords once a total of 10 unique keywords are found, optimizing the operation by limiting the number of keywords processed.

```
# Async function to extract keywords from articles
async def extract_keywords_from_links(links, api_key):
    async with aiohttp.ClientSession() as session:
        tasks = [get_industry_from_diffbot(session, link, api_key) for link in links]
        results = await asyncio.gather(*tasks)

        all_keywords = set()
        for article_data in results:
            if article_data and 'tags' in article_data:
                all_keywords.update([tag['label'] for tag in article_data['tags'][:10] if 'label' in tag])
                if len(all_keywords) >= 10:
                    break

        return list(all_keywords)[:10] # Return up to ten unique keywords
```

# OUR PROCESS PART 2

- Vectorizes a plugged in mission statement and priorly found keywords to run a cosine similarity test, in order to find the most applicable keywords.
- Plugs in each output of industry, links, keywords, and metadata into our MongoDB database, in our to output the values out of the MongoDB database if there is a repeat submission or a repeat industry.
  - The database includes these features and uses industry as the label.

# Keyword Search Optimization #2

- In our previous extract-keywords-from-links function, we gathered a substantial list of general keywords related to the submitted website.
- To recommend the most relevant keywords, we first ask the user for the website's mission statement. We then vectorize both the initially suggested keywords and the mission statement. By running a cosine similarity test, we identify and output the keywords that most closely align with the mission statement, ensuring they are the most applicable to the website.

```
# Function to get the mission statement from user
def get_company_mission():
    mission = input("Enter the company's mission statement: ")
    return mission

# Function to vectorize keywords
def vectorize_keywords(keywords):
    vectorizer = TfidfVectorizer(stop_words='english')
    vectors = vectorizer.fit_transform(keywords)
    return vectors, vectorizer

# Function to find the most relevant keywords to the mission
def find_relevant_keywords(mission, keywords):
    # Vectorize the mission and keywords
    vectors, vectorizer = vectorize_keywords(keywords)
    mission_vector = vectorizer.transform([mission])

    # Calculate cosine similarity
    cos_similarities = cosine_similarity(mission_vector, vectors).flatten()

    # Find the indices of the ten most relevant keywords
    ten_most_relevant = np.argsort(cos_similarities)[-10:]

    # Extract and return the corresponding keywords
    relevant_keywords = [keywords[index] for index in ten_most_relevant]
    return relevant_keywords
```

# How do you make your jokes rank high on Google? You give them meta-gags!

- The function iterates through a list of links, fetching article data for each link using a predefined function ('get\_industry\_from\_diffbot'). It then compiles a list of dictionaries, with each dictionary containing metadata from one article, including the article's URL, title, author, and publication date.
- For each piece of metadata (title, author, date), the function uses the 'dict.get' method to safely retrieve the value if it exists or provide a default string (e.g., 'No title available', 'Author unknown', 'No date available') if the data is missing. This ensures the function handles missing information gracefully and maintains a consistent data structure for all articles processed.

```
def extract_metadata_from_links(links):  
    metadata_list = [] # List to hold metadata dictionaries for each link  
    for link in links:  
        article_data = get_industry_from_diffbot(link) # Reusing the function to get the article data  
        if article_data:  
            # Create a dictionary to store relevant metadata from the article  
            metadata = {  
                'url': link,  
                'title': article_data.get('title', 'No title available'),  
                'author': article_data.get('author', 'Author unknown'),  
                'date': article_data.get('date', 'No date available')  
            }  
            metadata_list.append(metadata)  
    return metadata_list
```

# MongoDB Database to Command Back Implementation

- We created a MongoDB database to hold industries and their correlated keywords.
- If a user runs a website that already has been inputted, the mongoDB database will output the best applicable keywords that are already in the database, and also run asynchronously to increase the number of keywords in the database, ensuring improved keyword output by the program.

What did the MongoDB database say to the SQL Database?  
“Relaxxxxx I’m non-relational!”

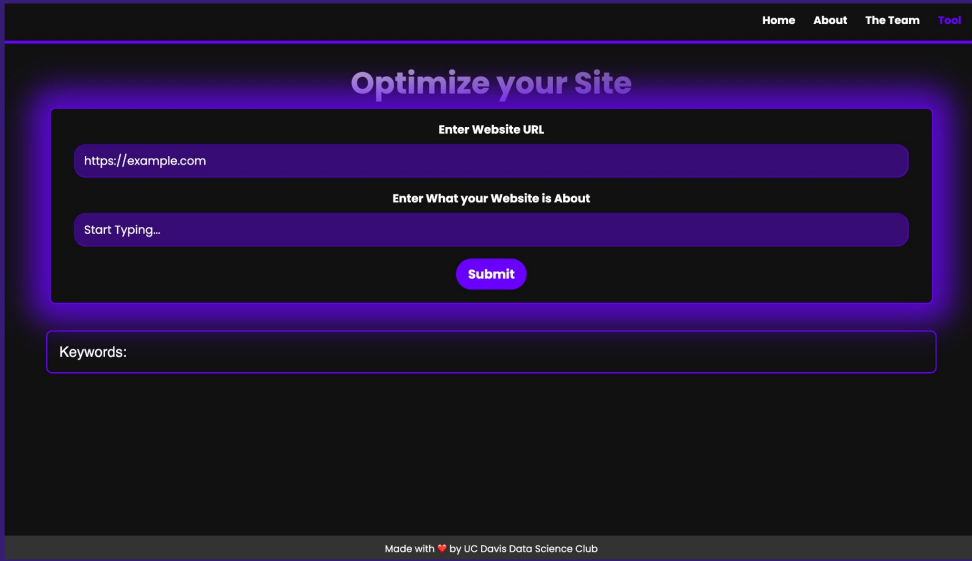


# Speed Optimization

- We utilized `aiohttp` for concurrent network requests to fetch data from multiple URLs simultaneously instead of sequentially to implement concurrent HTTP requests.
- We limited redundant API Calls by cache data retrieved from previous API calls to prevent fetching the same information multiple times.
- We reduced data processing by streamlining data processing by stopping once necessary conditions are met (e.g., collecting enough industries or keywords) to avoid unnecessary computations.
- We utilized efficient data structures, such as sets for faster membership checks and uniqueness, to enhance performance to optimize.
- We batched API requests and used bulk endpoints to retrieve multiple data points in a single call, thereby reducing the total number of HTTP requests.

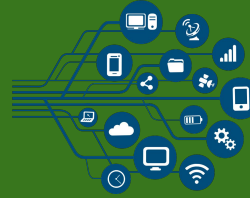
# Front-End Development

- We built a website to hold the tool that requests the user for a URL as well as the statement mission.
- The tool outputs the most applicable industry, keywords, metadata that ensure that the website improves its performance in any given search engine ranking system.



The screenshot displays a web application interface with a dark theme. At the top, a navigation bar contains links for 'Home', 'About', 'The Team', and 'Tool'. The main heading is 'Optimize your Site'. Below this, there are two input fields: 'Enter Website URL' with the text 'https://example.com' and 'Enter What your Website is About' with the placeholder 'Start Typing...'. A 'Submit' button is positioned below the second input field. At the bottom of the form area, there is a 'Keywords:' label and an empty input field. The footer of the page states 'Made with ❤️ by UC Davis Data Science Club'.

# FUTURE APPLICATIONS



- Futuristically, if a user inputs a website with an overlapping industry of one that is already on our database,
  - It sends the pre-saved output to our user
  - It asynchronously runs our web scraper code again to build on our database.
- As our database expands,
  - Our data will be more readily available
  - Our tool will become more efficient due to less need for web scraping.
  - Our tool will become increasingly accurate, as a result of a larger dataset, and vectorized search.



The background is a solid dark green. It is decorated with an abstract pattern of small squares in white, light blue, and light green, and thin white vertical lines of varying lengths, creating a modern, geometric aesthetic.

# PROJECT DEMONSTRATION

The background is a solid dark green. It is decorated with a pattern of small squares and thin vertical lines. The squares are in three colors: pink, cyan, and orange. Some squares are solid, while others are hollow. The vertical lines are thin and white, extending from the top of the slide to various heights. The text 'THANK YOU!' is centered in a large, white, sans-serif font.

# THANK YOU!