Analysis of Repetition in Teaching

Authors: Nastaran Mesgari

1 Exclusive Summery

In order to find duplications and raise the standard of the curriculum overall, this project focuses on comparing the content of two courses—AI and visualization. This was accomplished by taking a methodical approach. Two courses were chosen: one on artificial intelligence, which covered subjects like data models, algorithms, and machine learning; the other on visualization, which placed an emphasis on graphical data representation, storytelling, and insight generation. For each course, a set of domain-specific keywords was created in order to find and quantify content overlaps. “AI,” “machine,” “learning,” “intelligence,” “algorithm,” and “data” were among the keywords related to AI, while “story,” “narrative,” “visualization,” “insights,” and “emotion” were among the keywords related to visualization.

To guarantee accurate analysis, the course materials were preprocessed, which included eliminating unnecessary text, changing to lowercase, and transforming the data into formats that are easier to handle.**82 AI-related keywords and 27 visualization-related keywords** were found in the AI course, according to a keyword frequency analysis. there were 216 keywords related to visualization and 123 related to AI in the visualization course. similarity score of **0.2433** was obtained from this overlap, suggesting a moderate degree of redundancy between the two courses. To further show the distribution of keyword and give a visual depiction of the most common terms in the materials, word clouds were created for both courses.

The results indicate that although the courses retain their distinctt , there are significant conceptual overlaps, especially in fields pertaining to data and model application. This overlap suggests possible duplications that, if resolved, could improve the uniqueness and efficacy of each course. In order to find more extensive redundancy patterns throughout the curriculuom, the analysis could be extended to include more courses as a subsequent step. To learn more about content overlaps, advanced Natural Language Processing (NLP) method as like as topic modeling and similarity analysis could be used. These results could guide focused content optimization tacti, guaranteeing a distinct distinction between courses and enhancing students’ overall educational experience. To improve course content, reduce duplication, and develop a more coherent curriculum, cooperation with instructors is advised.

This project lays the groundwork for future initiatives to maximize academic programs by highlighting the significance of assessing instructional materials for alignment and redundancy. Universities can improve learning outcomes, encourage innovation, and preserve student learning by making sure that courses are streamlined and complementary.The insights gained from this analysis are instrumental in achieving these goals and improving the quality of teaching and learning.

2 Instroduction

In academic settings, the quality and structure of course content significantly impact students’ learning experiences and outcomes. With the growing importance of fields like Artificial Intelligence (AI) and Data Visualization, the need for well-structured and non-redundant educational programs has become increasingly critical. Redundant content, while sometimes unavoidable, can lead to inefficiencies in learning, reduce engagement, and limit students’ exposure to new concepts. Addressing such overlaps is essential for optimizing course offerings and ensuring students receive a comprehensive yet streamlined education.

This project aims to analyze and evaluate the extent of content similarities between two courses—one focusing on AI and the other on Visualization. These domains, while distinct, often intersect in their discussion of data-driven decision-making and the application of models to real-world problems. By systematically identifying overlaps, the project seeks to highlight areas of potential redundancy and offer solutions for improving course alignment.

The analysis is centered on identifying keywords representative of the core themes in each course. Preprocessing techniques are applied to refine the course materials for accurate assessment, followed by similarity metrics to quantify the degree of overlap. The results are visualized through tools such as Word Clouds, providing a clear picture of the shared and unique elements of the courses.

This initiative not only supports the academic goals of content optimization but also fosters collaboration among instructors. By aligning course objectives and minimizing redundancies, educators can offer more diverse and impactful learning experiences. This project serves as a stepping stone for broader efforts to evaluate and enhance curriculum design, ultimately contributing to higher standards in education and better preparing students for the challenges of an ever-evolving professional landscape.

To carry out this project, a step-by-step approach was followed to properly analyze the content similarity between two courses related to “Artificial Intelligence” and “Data Visualization.” These steps included data identification, text processing, feature extraction, and similarity analysis. Below is a breakdown of each step:

1. Course Selection Initially, two courses related to “Artificial Intelligence” and “Data Visualization” were chosen as the subjects for the similarity analysis. For each of these courses, the corresponding text files were extracted and prepared for processing:

Course 1: AI2.docx (Content related to Artificial Intelligence) Course 2: Vir1.docx (Content related to Data Visualization)

2. Definition of Key Terms Next, a set of key terms related to the topics of each course was defined. These terms were used to identify and compare the similarities between the courses:

For the AI course: {‘ai’, ‘machine’, ‘learning’, ‘intelligence’, ‘algorithm’, ‘data’, ‘model’} For the Data Visualization course: {‘story’, ‘narrative’, ‘data’, ‘visualization’, ‘insights’, ‘message’, ‘emotion’}

3. Text Processing and Cleaning The content of both text files was processed to remove any noise and irrelevant data. This step involved converting the text into a standard format, making it ready for analysis to ensure the results were accurate.

4. Keyword Analysis At this stage, the number of key terms found in each course was counted. This count served as a metric for comparing the similarities and differences:

In Course 1: 82 AI-related terms and 27 Data Visualization-related terms. In Course 2: 123 AI-related terms and 216 Data Visualization-related terms.

5. Similarity Calculation Using basic formulas, the similarity between the two courses was calculated. One of the metrics used was the comparison of the frequency of key terms in each course, resulting in a similarity score of 0.2433. This indicates a meaningful degree of similarity between the two courses, suggesting there may be overlap in the teaching of certain concepts.

6. Word Cloud Generation A Word Cloud was created for each course, visually representing the distribution of key terms in their content. This visualization aids in better understanding the similarities and differences between the courses

at the first I import my necessary library

▶ Code

3 Text Processing and Cleaning

In any text analysis task, the first crucial step is text processing and cleaning. This stage ensures that the raw textual data is ready for further analysis, such as keyword extraction, sentiment analysis, or similarity detection. During text processing, we remove unwanted characters, irrelevant information, and perform normalization, so the text can be analyzed in a consistent and structured manner.

In this project, text processing and cleaning involved extracting content from .docx files, removing unnecessary numerical data from filenames, and preparing the content by structuring it into a usable format for analysis. This process is vital to ensure that only the relevant text data is considered in subsequent steps of the analysis.

4 Find Similarity

▶ Code

```
Filename: AI2.docx - Label: AI
Content:
OK. So that's a little bit the results. OK, so we are somewhere between intermediate and beginners.
Great that you feel confident that you might learn the AI concepts. That's very good to know, and o...
----------------------------------------
Filename: Vir1.docx - Label: Vir
Content:
Mine too, but last week was I was in Oxford for a seminar and had to prepare the courses.
So this was quite stressful and it seems this semester for the new students is even more confusion with getti...
----------------------------------------
```

In this stage, we performed several preprocessing steps to prepare the text data for similarity analysis. The preprocessing steps we implemented include: **(Similarity between AI2.docx and Vir1.docx: 0.8910 and it is too much)**

Converting text to lowercase: This step ensured that text comparison was case-insensitive.

Removing numbers: Digits were removed to focus solely on the textual content.

Removing punctuation: This helped in standardizing the text and reducing noise.

While these preprocessing steps were helpful, we identified areas where improvements were necessary to achieve a more accurate similarity assessment. To enhance our approach, we implemented the following additional steps:

Removing stop words: Commonly used words like “the,” “and,” or “is,” which do not contribute significantly to the meaning, were removed to reduce their impact on the similarity calculation.

Applying lemmatization or stemming: This step involved reducing words to their base or root form (e.g., “running” to “run”), allowing us to treat different forms of the same word as identical.

Focusing on key terms: We prioritized words related to the specific topics of interest (e.g., “AI,” “algorithm,” “model,” “visualization”) to limit the analysis to relevant content.

Handling repetitive sections: Repeated phrases or structural similarities across documents were addressed to prevent inflated similarity scores. Exploring semantic similarity: Advanced methods, such as using semantic models (e.g., BERT or Word2Vec), were considered to account for contextual meanings rather than solely relying on word matching.

These additional steps were crucial in refining the preprocessing process and ensuring that the similarity analysis yielded meaningful and accurate results.

▶ Code

```
Similarity between AI2.docx and Vir1.docx: 0.8910
```

The purpose of this code is to compare two text documents to identify **common words** and **common bigrams (sequences of two consecutive words)** between them. The code aims to provide a better understanding of the overlap and similarity between the two documents. The main steps and objectives are as follows:

1. **Reading and Preprocessing Files:**
   - Text files (in `.docx` format) are read from a specified directory.
   - The content of each file is extracted and preprocessed:
     - Converted to lowercase (case normalization).
     - Numbers and punctuation are removed.
   - The goal of preprocessing is to standardize the text for more accurate analysis.
2. **Tokenizing the Text:**
   - The text of each file is split into a list of words (tokens).
   - This step enables the identification of repeated and common words.
3. **Finding Common Words and Calculating Frequency:**
   - Common words between the two documents are identified using set intersection (`&`).
   - The frequency of each common word in both files is calculated and displayed.
4. **Identifying Common Bigrams (Two Consecutive Words):**
   - Each text is transformed into a sequence of bigrams (pairs of consecutive words).
   - Common bigrams between the two documents are extracted and displayed.
   - Examining bigrams helps identify recurring structures or phrases between the texts.

The code is designed to analyze the similarity between two text documents in terms of content and structure. The results can be used to evaluate thematic overlap, detect potential repetitions, or refine textual content.

4.1 Reasons for High Similarities:

If there are many common words or bigrams between the documents, it may be due to the following reasons: 1. **Similar Topics:** Both documents might cover the same subject matter. 2. **Repetitive Phrases:** Standardized phrases or templates may have been used in both files. 3. **Insufficient Preprocessing:** Non-essential words (e.g., “the,” “is”) were not removed, affecting the similarity results.

▶ Code

4.2 What This Code Shows:

This code applies **Latent Dirichlet Allocation (LDA)**, a topic modeling technique, to identify the underlying topics within the provided `.docx` files. The output, which lists the **top words for each topic**, reveals the key terms most strongly associated with each identified topic.

For example:

- **Topic 1:** [‘yeah’, ‘ok’, ‘just’, ‘think’, ‘right’, ‘data’, ‘like’, ‘maybe’, ‘bit’, ‘umm’]
- **Topic 2:** [‘thats’, ‘ai’, ‘things’, ‘probably’, ‘say’, ‘yeah’, ‘decision’, ‘like’, ‘maybe’, ‘ok’]

These words represent the **essence of the topics** extracted by the model from the provided documents.

4.3 Purpose and Use for Subsequent Steps:

1. **Understanding Textual Themes:**
   - The output helps us understand the primary themes or topics present in the documents. For instance:
     - **Topic 1** seems to focus on conversational terms, possibly indicating informal communication or brainstorming.
     - **Topic 2** includes terms like “AI” and “decision,” which might suggest discussions related to artificial intelligence and decision-making.
2. **Refining Text Processing Pipelines:**
   - By analyzing these topics, we can identify noise in the text (e.g., filler words like “yeah,” “maybe,” “ok,” and “just”).
   - This insight can guide further preprocessing, such as customizing the stop-word list to exclude irrelevant terms and focus on more meaningful content.
3. **Basis for Text Classification or Comparison:**
   - Extracted topics can serve as features for comparing documents, grouping similar texts, or even training classification models if labels are available.

4. **Improving Alignment With Previous Results:**
    - In earlier steps, we identified word frequencies, common words, and bigrams. This approach builds on those analyses by summarizing the **conceptual structure** of the text, helping to better understand similarities and differences between documents.

4.4 Why Use This After Previous Code:

- The previous steps analyzed **word-level overlap** (e.g., shared words and phrases). However, they did not provide a high-level understanding of the themes or topics.
- LDA adds a layer of abstraction by grouping related words into topics, giving a clearer picture of the documents' **semantic content**.
- This approach ensures we're not just comparing superficial textual similarities but also deeper thematic alignments, which is crucial for making informed decisions in downstream tasks.

this code helps identify the main topics in the documents, providing a thematic overview that can be used to refine the preprocessing pipeline and guide further analyses. but as you can see the resault is not good and we decided change the code and check the other ways.

Topic 1: ['yeah', 'ok', 'just', 'think', 'right', 'data', 'like', 'maybe', 'bit', 'umm'] Topic 2: ['thats', 'ai', 'things', 'probably', 'say', 'yeah', 'decision', 'like', 'maybe', 'ok']

▶ Code

```
Topic 1:
['yeah', 'ok', 'just', 'think', 'right', 'data', 'like', 'maybe', 'bit', 'umm']
Topic 2:
['thats', 'ai', 'things', 'probably', 'say', 'yeah', 'decision', 'like', 'maybe', 'ok']
```

4.5 Comparison Between the Two Codes

Both codes aim to extract **topics** from `.docx` files using **Latent Dirichlet Allocation (LDA)**, but there are significant differences in their **preprocessing steps** and the resulting outputs.

4.6 Key Differences:

1. **Custom Stopwords:**
    - **Previous Code:** Relied only on the built-in English stopword list provided by `CountVectorizer`.
    - **Current Code:** Adds a **custom stopword list** (`['yeah', 'ok', 'umm', 'just', 'like', 'bit', 'maybe', 'right', 'thats']`) to filter out common filler words that do not contribute to meaningful topics.

2. **Preprocessing:**
    - **Previous Code:**
        - Lowercased text.
        - Removed digits and punctuation.
    - **Current Code:**
        - Includes all previous preprocessing steps.
        - Additionally removes **custom filler words**, making the data cleaner and more topic-focused.

3. **Output Topics:**
    - **Previous Code:**
        - **Topic 1:** ['yeah', 'ok', 'just', 'think', 'right', 'data', 'like', 'maybe', 'bit', 'umm']
        - **Topic 2:** ['thats', 'ai', 'things', 'probably', 'say', 'yeah', 'decision', 'like', 'maybe', 'ok']
        - These topics contain numerous filler words (e.g., "yeah," "ok," "maybe") that obscure the main themes.
    - **Current Code:**
        - **Topic 1:** ['ai', 'things', 'probably', 'say', 'decision', 'kind', 'human', 'actually', 'basically', 'different']
        - **Topic 2:** ['think', 'data', 'lets', 'know', 'say', 'income', 'dont', 'course', 'really', 'want']
        - The removal of filler words results in more **coherent topics**, highlighting terms related to artificial intelligence, decision-making, and data.

4. **Clarity of Topics:**
    - The **current code** generates more focused topics, making them easier to interpret and align with the context of the documents.

4.7 Comparison of Results:

4.7.1 Previous Results:

- Contained many **irrelevant filler words**.
- Topic differentiation was less clear due to noise in the data.

4.7.2 Current Results:

- More meaningful and specific:
  - **Topic 1** emphasizes AI-related terms like “ai,” “decision,” and “different.”
  - **Topic 2** relates to discussions about data analysis, with terms like “data,” “income,” and “course.”
- The removal of custom stopwords improved **topic clarity**.

4.8 Why the Current Code Was Used After the Previous One:

1. **Improvement in Preprocessing:**
   - Observing the prevalence of filler words in the previous output indicated a need for additional filtering. Adding a **custom stopword list** addressed this issue.
2. **Enhancing Interpretability:**
   - By focusing on meaningful words, the current approach provides **clearer and more actionable insights** about the themes in the documents.
3. **Aligning Topics with Goals:**
   - The improved topics can be better leveraged for tasks like document comparison, classification, or deeper semantic analysis.

4.9 Use in Subsequent Steps:

- The clearer topics can now serve as:
  - **Features for clustering documents** into related groups.
  - **A guide to refine further preprocessing** for different datasets.
  - **Basis for thematic analysis** or comparisons between texts.

the current code improves on the previous one by eliminating noise, generating cleaner topics, and enabling more focused analyses.

▶ Code

```
Topic 1:
['ai', 'things', 'probably', 'say', 'decision', 'kind', 'human', 'actually', 'basically', 'different']
Topic 2:
['think', 'data', 'lets', 'know', 'say', 'income', 'dont', 'course', 'really', 'want']
```

4.10 Explanation of the Code:

This version introduces an **advanced preprocessing step** by incorporating **Named Entity Recognition (NER)** using SpaCy. This enhances the clarity of topics extracted by the Latent Dirichlet Allocation (LDA) model.

4.11 Key Features of the Code:

1. **Named Entity Recognition (NER) Filtering:**
   - The code uses SpaCy’s `en_core_web_sm` model to identify and remove named entities (e.g., names, places, dates, organizations) from the text.
   - By excluding these entities, the focus shifts to general terms and patterns, reducing noise caused by specific names or details.
2. **Stopword Customization:**
   - The code expands the custom stopword list to include more filler words such as `'say', 'lets', 'think', 'know', 'course', 'really', 'probably'`.
   - This refinement ensures that the resulting topics are not dominated by irrelevant or generic terms.
3. **Enhanced Preprocessing Pipeline:**
   - Converts text to lowercase.
   - Removes digits and punctuation.
   - Filters out named entities and stopwords.
4. **Topic Modeling:**
   - The vectorization step (`CountVectorizer`) converts the cleaned text into a **document-term matrix**.
   - **Latent Dirichlet Allocation (LDA)** is applied to extract two topics (`n_components=2`).
   - The top 10 terms for each topic are displayed.

4.12 Comparison with Previous Codes:

1. **Addition of NER:**
   - In previous versions, named entities (like “AI,” “human,” “decision”) were included in the topic extraction process. While these entities might occasionally be relevant, their overrepresentation could obscure broader patterns.
   - The current code removes such entities to focus on more general and **conceptual terms**.
2. **Improved Topic Clarity:**
   - By removing named entities, the extracted topics emphasize broader patterns instead of being skewed by document-specific details.
3. **Expanded Stopwords List:**
   - The expanded stopword list further reduces noise and highlights meaningful words.

4.13 Expected Results:

- **Topics from Previous Code:**
  - **Topic 1:** [‘ai’, ‘things’, ‘probably’, ‘say’, ‘decision’, ‘kind’, ‘human’, ‘actually’, ‘basically’, ‘different’]
  - **Topic 2:** [‘think’, ‘data’, ‘lets’, ‘know’, ‘say’, ‘income’, ‘dont’, ‘course’, ‘really’, ‘want’]
- **Topics from Current Code:**
  - The new topics are expected to:
    - Exclude specific names and entities.
    - Highlight key themes or terms related to the general context of the documents.
    - Example: [“decision-making,” “data analysis,” “technological impact”].

4.14 Advantages of This Code for Future Steps:

1. **Generalization:**
   - Topics derived from this process are likely to generalize better across different datasets since named entities and filler words are removed.
2. **Suitability for Downstream Tasks:**
   - The output can be used for:
     - **Document classification or clustering.**
     - **Thematic comparison** across files.
     - **Keyword extraction** for summarization.
3. **Focused Analysis:**
   - By eliminating unnecessary noise, this method lays a stronger foundation for deeper text analysis or comparison.

4.15 Why Was This Update Introduced?

- To address **entity-specific noise** in the previous results.
- To enable broader **generalization** and clearer topics by eliminating unnecessary details.
- To prepare the data for future analyses that require a **higher-level understanding** of document themes.

▶ Code

```
Topic 1:
['data', 'nt', 'income', 'things', 'people', 'want', 'time', 'look', 'lot', 'different']
Topic 2:
['ai', 'decision', 'based', 'aspects', 'language', 'database', 'autonomous', 'driving', 'intelligence', 'ultimately']
```

▶ Code

▶ Code

This code introduces a **comprehensive text preprocessing pipeline** which performs various text cleaning and preparation steps before further analysis. Here’s a detailed breakdown of what this code does:

4.16 Key Features of the Code:

1. **Text Cleaning with `text_cleaner` Function:**
   - **Remove `@` symbols:** Any word starting with `@` is removed, likely to exclude social media mentions.

- **Substitute non-alphabetic characters:** It replaces any non-alphabetic characters (like punctuation and special characters) with spaces.
- **Lowercasing:** All text is converted to lowercase to standardize it.
- **Remove extra spaces:** Extra spaces are collapsed into a single space, and leading/trailing spaces are removed.
- **Tokenization:** It breaks the text into individual words (tokens).
- **Remove digits and non-alphabetic words:** Any digits or non-alphabetic words are removed.
- **Remove stopwords:** The list of common English stopwords (like “the”, “and”, “is”) is excluded from the text.
- **Lemmatization:** Words are lemmatized (reduced to their base form), considering their part-of-speech (POS).

2. **Lemmatization with POS Tagging:**
   - Each word is tagged with its **Part of Speech (POS)**, and based on the POS, the correct lemmatization process is applied.
   - Words are lemmatized into their root form (e.g., “running” becomes “run”, “better” becomes “good”).

3. **Reading .docx Files:**
   - `read_docx` **function** reads the content of `.docx` files, extracting the text from all paragraphs and concatenating them.

4. **Iterating through Files in a Directory:**
   - The script iterates through all files in a given directory (`directory` variable), and for each `.docx` file, it reads and preprocesses the text.
   - The cleaned text is then stored in a list (`texts`), and filenames are stored in `filenames`.

5. **Displaying Cleaned Text:**
   - The cleaned text for each file is printed for verification, so you can visually check how the text has been processed.

4.17 Function Breakdown:

- `text_cleaner` **function** performs the core preprocessing:
  - Removes words starting with `@`.
  - Strips out non-alphabetical characters.
  - Tokenizes, removes digits, filters valid words, and removes stopwords.
  - Lemmatizes the remaining words and reassembles the cleaned text.
- `lemmatize` **function** uses POS tagging to apply the correct lemmatization based on the word’s part of speech (noun, verb, adjective, etc.).

4.18 Expected Output:

For each `.docx` file in the specified directory, the program will output the cleaned text. The cleaned text will have: - No special characters, digits, or stopwords. - All words lemmatized to their base forms (e.g., “running” becomes “run”). - Words such as “better” would be lemmatized to “good.”

Here’s an example of what the output might look like:

```
Cleaned text from document1.docx:
this study focus on analysis of data from various sensor to predict future trends in health care.
---------------------------------------
Cleaned text from document2.docx:
ai technology is becoming increasingly important in decision making across multiple industries.
---------------------------------------
```

4.19 Why This Code is Useful:

- **Data Preprocessing for NLP Models:** The cleaned text is ready for further natural language processing tasks such as topic modeling, sentiment analysis, or text classification.
- **Consistent Format:** By removing stopwords, digits, special characters, and applying lemmatization, the text becomes standardized, reducing noise and improving the accuracy of downstream models.
- **Scalability:** This method works for any number of `.docx` files, and it’s easy to modify if you need to preprocess other types of files.

4.20 Potential Improvements/Modifications:

- **Custom Stopwords List:** You might want to expand or modify the stopword list to better suit your specific dataset.
- **POS Tagging Improvement:** You can fine-tune the POS tagging process or integrate a more sophisticated lemmatizer if needed for specialized vocabularies.

▶ Code

▶ Code

▶ Code

▶ Code

The code I've provided is well-structured for cleaning and preprocessing text data, particularly from `.docx` files. It removes unnecessary parts of the text (e.g., session details, timestamps), performs tokenization, and filters out stop words, digits, meaningless words, and duplicates. Additionally, it performs lemmatization to ensure that the words are reduced to themy base forms.

Here's a breakdown of how the code works:

1. **Text Cleaning (Regex Replacements):**
   - It removes specific unwanted patterns (e.g., class sessions, AM/PM timestamps) using regular expressions (`re.sub`).
   - Non-alphabetic characters are replaced with spaces (`re.sub(r'[^0-9a-zA-Z\s]', ' ', contents)`), and multiple spaces are reduced to a single space.

2. **Tokenization and Filtering:**
   - Tokenizes the text into words using `nltk.word_tokenize`.
   - Filters out digits, invalid words, and stop words.
   - It also removes custom meaningless words (`4bs`, `f`, `xh`) and patterns like alphanumeric combinations (e.g., "ai4bs").

3. **Lemmatization:**
   - Uses `WordNetLemmatizer` from NLTK to convert words to themy base form (e.g., "running" to "run").

4. **File Handling:**
   - Reads `.docx` files using the `docx` library.
   - For each file, the `text_cleaner` function is applied to preprocess the content.

5. **Dmyectory Traversal:**
   - Reads all `.docx` files from a specified dmyectory, processes them, and stores the cleaned content in a list.

Finally, the cleaned text from each file is printed for verification.

4.21 Potential Improvements:

- **Error Handling:** It might be useful to add error handling for cases where the `.docx` files cannot be read or processed correctly.
- **Efficiency:** If the dmyectory contains many files, I could consider processing the files in parallel or batch processing to speed up execution.

the code works as expected, I see the cleaned and processed text output from each `.docx` file in my dmyectory.

▶ Code

To begin, I decided to use the textual data from the .docx files. I started by reading the files and preprocessing the text using the preprocess_text function. The goal in this step was to clean up the text by removing noise such as punctuation or irrelevant words, converting the text into a simpler form that would be more suitable for analytical models.

After preprocessing the texts, I applied Topic Modeling to analyze the data. For this task, I used the Latent Dirichlet Allocation (LDA) model, which helped me identify different topics within the texts. To begin, I converted the texts into a document-term matrix using CountVectorizer, where each row represented a document and each column represented a word.

To ensure that the model accurately identified topics, I set the number of topics to 2 (this number can be adjusted based on the data). I selected this number because I wanted the model to extract two primary categories of concepts.

After running the model, I ended up with the following topics:

Topic 1:

['data', 'nt', 'income', 'things', 'people', 'want', 'time', 'look', 'lot', 'different'] Topic 1 is related to concepts such as "data," "income," and "time." It seems to refer to the analysis of data and its use in various contexts.

Topic 2:

['ai', 'decision', 'based', 'aspects', 'language', 'database', 'autonomous', 'driving', 'intelligence', 'ultimately'] Topic 2 focuses on concepts like "artificial intelligence," "decision-making," and "autonomous driving." This topic likely relates to applications of AI and related technologies.

Initially, I encountered some issues with the previous code, so I decided to implement this new approach to preprocessing and analyzing the texts. With this new method, I was able to identify the main topics present in the data and obtain meaningful results that aid in a better understanding of the texts.

▶ Code

```
Topic 1:
['data', 'nt', 'income', 'things', 'people', 'want', 'time', 'look', 'lot', 'different']
Topic 2:
['ai', 'decision', 'based', 'aspects', 'language', 'database', 'autonomous', 'driving', 'intelligence', 'ultimately']
```

Here's how I arrived at the difference in similarity scores between 89% and 47%, and the reasoning behind the process:

Initially, I started by reading and preprocessing the text data from all the .docx files. The goal was to clean the data, removing unwanted content (like session details or timestamps) and irrelevant words. This is where I focused on eliminating stop words, non-alphabetic characters, and some meaningless words like "4bs" or "f." I also performed lemmatization to normalize the words, ensuring that they appeared in their base forms (e.g., changing "running" to "run").

Once the preprocessing was done, I used TF-IDF (Term Frequency-Inverse Document Frequency), which is a method for transforming the text data into numerical vectors. This vectorization allows us to represent the documents in a way that makes it easier to calculate similarities between them.

For the cosine similarity step, I compared each document's TF-IDF vector to the others. Cosine similarity measures how close two vectors are, meaning it looks at how similar the content is between the documents. The score ranges from 0 (completely different) to 1 (identical).

For instance, I initially got a similarity score of 89% between two documents (let's call them AI2.docx and Vir1.docx). However, when revisiting the comparison later, I found a lower similarity score of 47%. This difference can be attributed to several factors:

Preprocessing: During the text cleaning process, some important words might have been removed, or the lemmatization might have altered the meaning slightly, which could have affected the final similarity scores. It's possible that in one iteration, the preprocessing steps removed more useful words, leading to a reduced similarity score.

TF-IDF Vectorization: The TF-IDF transformation is sensitive to the presence of rare terms and their importance in a document. If two documents have more unique terms that are not shared between them, the similarity score will be lower. It's likely that in one of the calculations, the key terms that defined the similarity between the documents were weighted differently, resulting in a significant drop.

Document Content: If the content of the documents is slightly varied, the similarity score will reflect that. For example, documents with highly overlapping words (like "ai," "data," "income," "decision") would have a higher cosine similarity, but if one document introduced more specific or different words, the similarity score could decrease.

Ranking of Keywords: The top keywords also play a crucial role. The keywords for AI2.docx include terms like "ai," "decision," and "probably," which are related to decision-making and artificial intelligence. On the other hand, the keywords for Vir1.docx focus more on terms like "yeah," "think," and "income," which suggest that the documents discuss slightly different topics. This difference in keyword relevance likely contributed to the observed drop in similarity between the two documents.

So, while the first result was 89%, this second calculation of 47% reflects a more accurate and nuanced similarity, taking into account the impact of preprocessing, vectorization, and the actual content of the documents.

▶ Code

```
Similarity between 'AI2.docx' and 'Vir1.docx': 0.4787

Top keywords in 'AI2.docx': ai, decision, probably, say, yeah, like, maybe, kind, human, actually

Top keywords in 'Vir1.docx': yeah, think, right, data, income, like, maybe, bit, let, say
```

4.22 Key Steps Taken:

1. **Text Preprocessing**:
   - First, I read the content of each `.docx` file using the `read_docx` function and extracted the text.
   - Then, I cleaned the extracted text by removing non-alphabetic characters and converting everything to lowercase, so that the focus was only on words without being affected by case sensitivity or non-alphabetical symbols.
   - After cleaning, I split the text into individual words using the `split()` function.
2. **Counting Keywords**:
   - I defined two sets of keywords: one for **AI** (`keywords_ai`) and one for **vir** (`keywords_vir`).
   - For each word in the cleaned text, I counted how many times words from each keyword set (AI and vir) appeared. This helped me measure the presence of AI-related and vir-related terms in each document.

4.23 Results:

- **File: 'AI2.docx'**
  - AI Keywords: 82
  - Vir Keywords: 29

  The `AI2.docx` file contains 82 occurrences of AI-related keywords and 29 occurrences of vir-related keywords. This indicates that the document focuses more on AI topics but also touches on vir-related terms.
- **File: 'Vir1.docx'**

- AI Keywords: 123
- Vir Keywords: 123

In the case of the `Vir1.docx` file, both AI and vir-related keywords appear equally, with 123 occurrences for each. This indicates that the document equally addresses both AI and vir topics.

4.24 How I Reached This:

1. **Text Extraction**: I started by extracting the content of the `.docx` files using the `python-docx` library.
2. **Text Cleaning**: Then, I cleaned the text by removing unnecessary characters and splitting it into words.
3. **Keyword Comparison**: Finally, I compared the content of each document with the keyword sets for AI and vir, counting how many times each set of keywords appeared.

This method helps me quickly identify which topic—AI or vir—is more prevalent in each document and where the primary focus lies.

▶ Code

```
File: 'AI2.docx', AI Keywords: 82, vir Keywords: 29
File: 'Vir1.docx', AI Keywords: 123, vir Keywords: 123
```

In this code, you are calculating the similarity between documents based on the occurrence of specialized keywords related to **AI** and **vir** (visualization and storytelling) topics.

4.25 Explanation of the Code:

1. **Text Preprocessing**:
   - You read each `.docx` file and clean the text by removing non-alphabetical characters and converting the text to lowercase.
   - The text is then split into words for analysis.
2. **Keyword Matching**:
   - You define two sets of keywords:
     - **Keywords related to AI**: `keywords_ai`
     - **Keywords related to vir**: `keywords_vir`
   - For each document pair, you count the number of AI and vir keywords that appear in both documents.
3. **Similarity Calculation**:
   - For each pair of documents, you calculate the number of shared AI and vir keywords.
   - You calculate the **similarity score** as the ratio of the total number of shared keywords (AI and vir) to the total number of keywords (AI + vir) in both documents.

4.26 How Similarity is Calculated:

1. **Keyword Counts**:
   - For each document, you count how many times AI and vir keywords appear.
2. **Shared Keywords**:
   - The similarity between two documents is calculated based on the **shared AI** and **shared vir** keywords. The number of shared keywords is the **minimum count** of a keyword in both documents.

4.27 Result:

- The similarity score between **'AI2.docx'** and **'Vir1.docx'** is **0.2433**.
  - This means that there is a 24.33% overlap in the AI and vir-related content of the two documents based on the defined keywords.

4.28 How This Result is Reached:

- **Document 1 ('AI2.docx')**: Has 82 occurrences of AI-related keywords and 29 occurrences of vir-related keywords.
- **Document 2 ('Vir1.docx')**: Has 123 occurrences of AI-related keywords and 123 occurrences of vir-related keywords.
- After counting and comparing the shared AI and vir keywords, the similarity score was computed as **0.2433**, indicating a low but significant overlap in the terms related to both AI and vir between these two documents.

▶ Code

```
Similarity between 'AI2.docx' and 'Vir1.docx': 0.2433
```

5 Conclusion:

In this analysis, you have examined two distinct fields: **Artificial Intelligence (AI)** and **Data Visualization** using specialized keywords for each domain. Through the extraction and processing of keywords, you have generated word clouds that highlight the distinctions and similarities between these two fields.

- **Artificial Intelligence (AI)** focuses more on technology, algorithms, and models, with keywords like “ai”, “learning”, “machine”, “intelligence”, and “algorithm”.
- **Data Visualization** emphasizes conveying information and storytelling through data, with keywords such as “message”, “visualization”, “context”, “story”, “audience”, “insights”, and “experience”.

As a result, you have successfully highlighted the differences between these two domains through their respective keywords, providing a better understanding of both fields and their interconnections.

6 Suggestions for Next Steps:

1. **Expand the Keyword Sets**:
    - To enhance the accuracy of your analysis, consider expanding the keyword sets for both domains. For example:
        - In the AI domain, you could add terms like “deep learning”, “neural networks”, “automation”, and “big data”.
        - In the Visualization domain, terms like “interactive”, “graph”, “dashboard”, “chart”, and “storytelling” could be included.

    Expanding these keywords will allow for more refined analyses and provide deeper insights into similarities and differences.

2. **Use More Advanced Models for Semantic Analysis**:
    - Simple keyword-based approaches are effective, but for greater accuracy, consider utilizing more advanced **Natural Language Processing (NLP)** models. Techniques such as **TF-IDF (Term Frequency-Inverse Document Frequency)** or **Word2Vec** can help you capture semantic and conceptual similarities between texts more precisely.

3. **Comparative Analysis and Clustering**:
    - For more detailed comparison and grouping of similar texts, consider using **clustering techniques** like **K-means** or **Hierarchical Clustering**. These methods can help you group similar texts together and provide a better understanding of relationships between them.

4. **Explore Relationships Between Domains**:
    - If you’re interested in deeper insights into the relationships between these two fields, you could perform **correlation analysis**. This will allow you to explore how concepts and keywords from one domain may appear in the other and how these domains might influence each other.

5. **Apply Machine Learning Models**:
    - For a more sophisticated comparison of texts, you could apply **machine learning models** like **neural networks** to analyze the texts. These models can process the data more deeply and uncover complex relationships between texts.

6.1 Summary:

You have laid a solid foundation for comparing the **AI** and **Data Visualization** domains based on their keywords. For the next steps, it is suggested that you expand your keyword sets, explore more advanced NLP and machine learning models, and conduct deeper semantic and comparative analyses to achieve more comprehensive and meaningful results.