Sorting your climate change-related indicators!

Lucie Kuczynski - email

2025-06-10

Warning

The script is expected to undergo significant updates in the future. One key improvement will be reducing the need for manual review of incorrect clusters by allowing the script to be re-run with adjusted settings—for example, applying stricter similarity thresholds.

To support this, the script will be reorganized. The first step will involve reformatting the data to match the required output structure. Once formatted, the data will be processed, and any clusters identified as incorrect can be reprocessed through additional rounds as needed, using refined parameters.

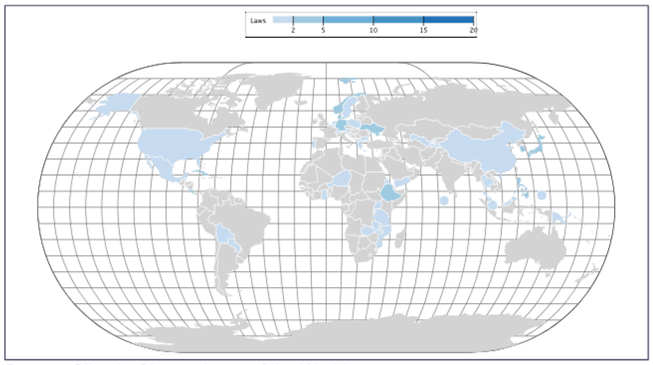
We've created a script that helps identify survey questions that are very similar to each other. It starts by tidying up the text of each question, then converts them into a format that makes comparisons easier. It checks how closely related the questions are, highlights the ones that are nearly identical, and groups similar ones together. This process helps reduce duplicate questions without losing any of the original information.

The script is built on sentence transformers, which offer a major advantage over traditional automated methods like edit-distance, TF–IDF, or topic modeling. While those approaches tend to focus on surface-level similarities or general themes, sentence transformers go deeper by learning to map semantically similar sentences (e.g., rewording, paraphrases) to nearly the same point in a dense vector space. This means that identifying paraphrased indicators becomes as simple as running a cosine similarity check between their embeddings. The result is a powerful, language-model-quality comparison - without the need to write any deep learning code yourself.

Why is it useful?

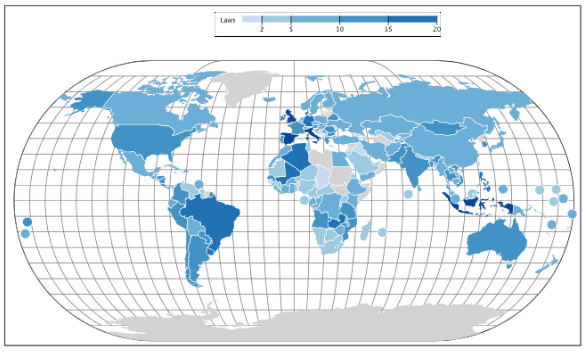
To address the challenges of climate change, various international frameworks, treaties, and conventions have been established. These initiatives aim not only to limit the progression of climate change but also to support adaptation efforts and assess its impacts. They typically include sets of indicators (both quantitative and qualitative) to track progress toward specific goals and evaluate the effectiveness of related policies. Over time, environmental treaties have developed mechanisms that allow them to adapt and respond to emerging needs. For instance, between 2017 and 2019, the number of national policies and measures reported by Member States to curb greenhouse gas emissions rose by approximately 27% (EEA, 2019).

Figure 2. Climate legislation in 164 countries in 1997



Source: Climate Change Laws of the World

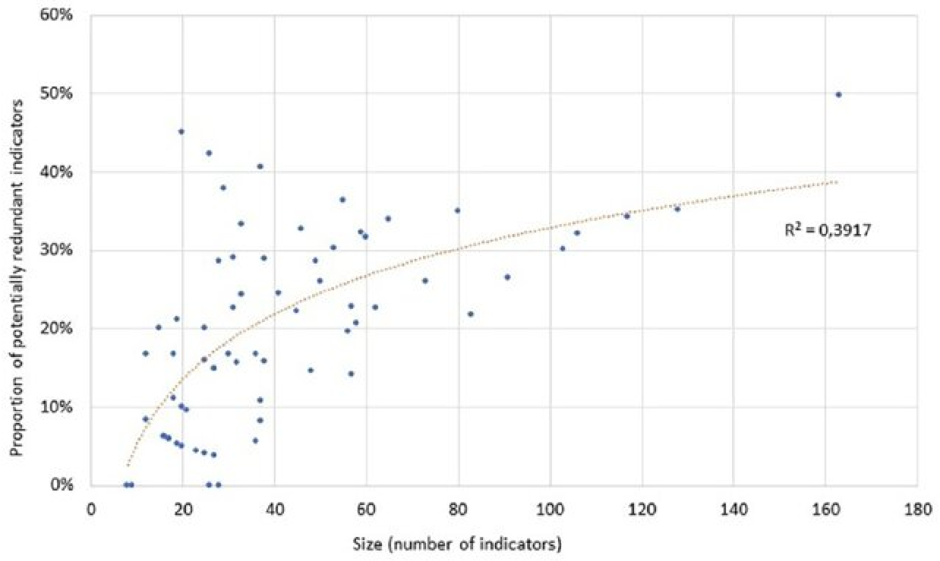
Figure 3. Climate legislation in 164 countries in 2017



Source: Climate Change Laws of the World

Online source, accessed June 2025

The downside of this otherwise adaptive approach is the growing number of indicators. As new frameworks emerge, they often introduce additional indicators that, over time, tend to overlap with existing ones. These older indicators are still maintained (e.g., they are familiar to stakeholders, their use preserves consistent time series) leading to redundancy across different monitoring systems.



Online source, accessed June 2025

A time-consuming approach to managing indicator redundancy is to manually review and classify each one based on perceived similarity. While this method doesn't require technical skills (just familiarity with the indicators) it becomes increasingly inefficient and error-prone as the number of indicators grows.

To streamline this process, we propose a script that performs an initial round of clustering, grouping similar indicators together. This allows users to review one cluster at a time, rather than comparing all indicators individually. For each cluster, the user can decide whether it makes sense and, if so, assign it a name; otherwise, the cluster can simply be tagged to be manually reviewed.

At the end of the process, a CSV file is generated, which the user can further edit manually. In the first trial of this script, applied to a set of ca. 350 climate change indicators, the user only needed to review fewer than 80 clusters. Of these, just 10 were found to be irrelevant and required manual adjustment.

How does it work?

The script is organized into seven steps, each corresponding to a specific section.

User settings, packages and import data (steps 1 to 3) The first three steps of the script are relatively straightforward. The user begins by defining key settings, with two particularly important parameters: `similarity_cutoff` and `min_cluster_size`.

The `similarity_cutoff` determines how closely two indicators must resemble each other to be grouped into the same cluster. This value ranges from 0 to 1, where lower values allow for more loosely related indicators to be clustered together, and higher values require greater similarity in phrasing.

The `min_cluster_size` sets the minimum number of indicators required for a cluster to be formed. In the example, this is set to 1, meaning even a single indicator that doesn't closely match any others can form its own cluster. If this value is increased (e.g., to 3), each cluster must contain at least that number of indicators, which means every indicator will be grouped with at least `min_cluster_size - 1` others (e.g. 2).

Finally, in the settings section, the user can modify the `file_path` to specify the location of the Excel sheet containing the questions.

Cleaning text (step 4) Because indicators may be reported with their own numbering systems, which can vary and be inconsistent across different frameworks, these numbers are automatically removed using the `clean_question` function. In addition, the function strips punctuation and collapses extra spaces. It intentionally retains stop-words and numbers within the definitions, as the semantic model can handle them effectively, and specific numbers or years can sometimes carry important meaning.

Transformer embeddings (step 5) This section handles the transformation of each survey question into a numerical format that a computer can work with. The first time the script runs, it downloads the MiniLM model (a compact, license-friendly tool trained on large amounts of text). MiniLM converts each question into a 384-dimensional vector, essentially a long list of numbers that captures the meaning of the sentence.

These vectors, known as embeddings, act like unique "fingerprints" for each question. They reflect not just the words used, but also their order, context, and nuances like negation or synonyms. For instance, this approach makes it possible to clearly distinguish between similar-sounding phrases like "proportion of energy from renewable sources" and "proportion of energy not from renewable sources". The embedding captures the presence of negation and other subtle differences in meaning, allowing the script to treat these as distinct indicators - embeddings provide a way to represent the meaning of text in a form that algorithms can analyze.

In practice, the script sends each question to MiniLM, which returns a vector (e.g., [0.12, -1.07, 0.35, . . . , -0.63]) with hundreds of values. Each row in the resulting `embeddings_matrix` corresponds to one question, now represented in a way that allows for meaningful comparison with others.

This step is currently parallelized to speed up computation. It automatically uses the number of available processor cores minus four, ensuring efficient performance while leaving some system resources free for other tasks.

Similarity in phrasing (step 6) To identify which questions are similar in meaning, the script computes a cosine similarity matrix. This involves first normalizing each question's embedding vector so that all vectors have unit length (i.e., resizing them to make them directly comparable). Once normalized, the script calculates the dot product between every pair of vectors using a fast, optimized function (`tcrossprod`), which avoids explicit loops and leverages BLAS acceleration for efficiency.

Cosine similarity measures how closely two vectors "point" in the same direction. If the similarity score is close to 1, it means the questions are likely expressing the same or very similar ideas. Scores near 0 or negative suggest the questions are unrelated or discussing entirely different topics.

Next, the script keeps only the upper-triangular portion of the similarity matrix to avoid duplicate comparisons (since similarity is symmetric), and exports a tidy table of potential duplicates to a file named `duplicate_pairs.csv`. A similarity threshold of 0.80 is used by default, which is generally a good cut-off for identifying paraphrases with MiniLM. If the results include too many false positives, this threshold can be changed (e.g., to 0.85 for stricter matching) in the settings section (Step 1) of the script.

In this context, false positives refer to pairs of survey questions that the script identifies as being very similar (i.e., potential duplicates or paraphrases), but which are not actually similar in meaning when reviewed by a human. For example, two questions might share a lot of the same words or structure, leading the model to assign them a high similarity score. However, they could be addressing different topics or have subtle but important differences in intent (e.g., "proportion of protected area of total area" and "proportion of protected area of land area"). These mismatches are considered false positives because the model "positively" flagged them as similar, but that classification turns out to be incorrect. Raising the similarity threshold (e.g., from 0.80 to 0.85) can help reduce the number of these false positives by requiring a stronger match before two questions are considered similar. However, setting the threshold too high can have the opposite effect: it may limit clustering to only those indicators that are phrased almost identically. This reduces the script's ability to detect meaningful paraphrases or slightly reworded versions of the same concept, potentially missing out on useful groupings. Finding the right balance is key to capturing both precision and nuance.

Cluster definition (step 7) Finally, the script transforms similarity scores into distances by subtracting the cosine similarity from 1. It then uses average-linkage clustering to group similar indicators. In this method, each indicator starts as its own cluster, and clusters are merged based on the average distance between all pairs of items across two clusters. This approach strikes a balance between being too lenient (like single-linkage) and too strict (like complete-linkage), resulting in more coherent and evenly shaped clusters. The clustering process produces a dendrogram, which is then cut at the same threshold used earlier to define duplicates. This ensures that within each cluster, every indicator is within the similarity threshold of at least one other member.

The script generates an output file called `duplicate_clusters.csv`, where each entry in the `cluster` column corresponds to a group of similar indicators. Currently, the `min_cluster_size` is set to 1, meaning that even a single indicator can form its own cluster if no close matches are found. These single-indicator clusters are retained as-is. When a cluster contains multiple indicators, the user is prompted to define a new, representative indicator that best captures the shared meaning of the group. This step not only helps consolidate similar content but also provides an opportunity to identify and flag false positives—cases where indicators were grouped together incorrectly. These can be tagged (e.g., as `not_a_cluster`) for further review or exclusion in later processing.

Currently, the final formatting of the data—such as identifying which frameworks are represented in each cluster based on the sources of the original indicators and any previously established links—takes place at this stage. However, future versions of the script will move this formatting to the beginning, allowing the similarity and clustering steps to be repeated more easily with different thresholds.

Why doing it this way?

When choosing MiniLM with cosine similarity for identifying duplicate or similar survey questions, the decision is based on both performance and practicality. Traditional methods like exact duplicate detection (e.g., using `duplicated()`) are limited to catching only byte-for-byte identical entries, missing any reworded or paraphrased versions. String distance metrics such as Levenshtein or Jaro–Winkler are only effective for very minor edits—like pluralization—and fail when sentence structure changes or synonyms are used.

Other approaches like TF-IDF with cosine similarity or topic modeling techniques (e.g., LDA or LSA) can capture general themes but fall short in understanding sentence-level meaning. They often misinterpret negations or subtle shifts in phrasing, and require extensive tuning, especially for short texts. While heavier models like full BERT embeddings offer slightly better accuracy, they come with significant computational costs. MiniLM, by contrast, offers a strong balance: it's lightweight, fast, and accurate enough for most practical use cases.

In terms of performance, MiniLM can process around 1,000 short questions per second on a standard laptop CPU. The cosine similarity matrix is computed efficiently using BLAS, and even for a few thousand questions, memory usage remains modest (i.e., about 4.4 MB for 3,000 questions). This efficiency allows the script to remain in base R, using only three external CRAN packages, while still delivering results comparable to deep learning models.

For users with compatible hardware, the script can be further optimized. Settings like `batch_size` can be adjusted to maximize GPU usage without exceeding memory limits, and enabling `via_parallel = TRUE` allows sentence tokenization and batching to run in parallel across R workers. Additionally, embeddings can be cached, so previously processed questions don't need to be re-embedded in future runs.

Altogether, this setup provides a powerful, efficient, and flexible solution for identifying and managing duplicate or similar survey indicators—without requiring deep technical expertise or heavy infrastructure.

Please note that this script is still under development, and any constructive feedback is more than welcome!