Okay, let's tackle this problem. The user is having issues with their document analysis system where the percentages for original, AI-generated, and plagiarized content don't add up correctly. They mentioned getting impossible values like 93% original, 99% AI, and 10% plagiarized, which clearly can't sum to 100. The goal is to adjust these calculations so they're mutually exclusive and accurate.

First, I need to understand how each score is calculated. The plagiarism score comes from comparing the text to existing documents using cosine similarity. The AI score is from a model that detects AI-generated content. The original score is just 100 minus the plagiarism score. But if AI and plagiarism are separate categories, they might overlap, leading to the sum exceeding 100.

The key issue is that the current setup treats these scores as independent, but they should be parts of a whole. For example, a sentence could be both plagiarized and AI-generated, but it shouldn't count twice. So, the scores need to be adjusted to ensure they don't overlap and sum up correctly.

I'll start by revising the AI detection logic. Instead of using the raw score from the model, I'll normalize it based on the portions of text not already flagged as plagiarized. That way, AI detection only considers the remaining text after accounting for plagiarism.

Next, the original content score should be what's left after subtracting both AI and plagiarism percentages. This ensures that all three categories sum to 100%.

I'll need to modify the `analyze_text` function to not only find plagiarized sections but also exclude those sections when calculating the AI score. Similarly, the AI detection should ignore already plagiarized parts. Then, the original score is simply 100% minus both AI and plagiarism scores.

Another consideration is the accuracy of the AI model. The user is using 'Hello-SimpleAI/chatgpt-detector-roberta', which might need calibration. Adding a confidence threshold and ensuring the model's output is scaled correctly will help in making the AI score more reliable.

I'll also adjust the response structure to include all three scores explicitly, ensuring they are calculated in a way that they are mutually exclusive and add up to 100%. This might involve passing both plagiarism and AI results through each function so that each can account for the other's findings.

Finally, I'll test the revised functions with sample texts to ensure the percentages make sense and the highlights don't overlap. This includes checking edge cases like completely original content, fully AI-generated, and fully plagiarized documents to verify the scores behave as expected.

Okay, I need to help the user resolve the "Found array with 0 sample(s)" error in their Django application. Let's see what's going on here.

First, the error message mentions that an empty array is being passed to a function that expects at least one sample. This likely happens when the AI detection model (check_ai_probability) receives text that's too short or invalid after extraction.

Looking at the extract_text_from_file function, it's supposed to handle PDFs, DOCX, and TXT files. The user has some error handling here, but maybe there are cases where it's returning empty text or text that doesn't meet the model's requirements. For example, image-based PDFs might not extract any text, leading to empty strings.

In the AnalyzeDocumentView post method, after extracting the text, they're doing some validations: checking length, presence of word characters, etc. But perhaps there's a case where these checks pass (like a very short text with a few characters), but the AI model still can't process it, leading to the empty array error.

The check_ai_probability function uses a Hugging Face pipeline. If the input text is too short or doesn't contain enough features for the model, it might return an empty result, causing the error.

Possible steps to fix this:

1. Enhance Text Extraction: Ensure that the extract_text_from_file function reliably extracts text and handles edge cases like image-based PDFs more gracefully. Maybe add OCR as a fallback, but that's complex. For now, improve error messages.

2. Stricter Validation Before AI Processing: Even if the text passes the current validations (like length >=50), it might still be insufficient for the model. Add a check for minimum word count, not just character count.

3. Error Handling in AI Function: Wrap the AI detection call in a try-except block to catch any errors and return a default score if processing fails.

4. Logging: Add more detailed logging to track where the failure occurs—whether the text extraction is failing or the AI model is the issue.

Let me look at the user's current code. They have a check for len(text) < 50 and regex for word characters. But maybe the extracted text is exactly 50 characters but still invalid. Adding a check for at least 3 words could help.

In check_ai_probability, they should ensure that the text is valid before passing it to the model. Also, initializing the model once (singleton) instead of every call might help with performance and resource management.

They might also be missing the case where the model returns an empty result. Adding a check for the result's existence before accessing it would prevent errors.

So, modifying check_ai_probability to handle empty results, adding stricter validations in the view, and improving logging should resolve the issue.