# 🟡 A Story with Claude & Usage

## What this Notebook does

Elevate your Word document with AI-powered proof feeding that goes beyond standard spell-check. This notebook uses Anthropic's to scan your Word documents, catching subtle language and style errors that slip past Microsoft Word. Whether you're writing in English, German, Italian, or French, it preserves your intended meaning while polishing your prose to perfection—even for very large documents.

Elevate your Word document with AI-powered proof feeding that goes beyond standard spell-check.

This notebook also demoes as a doubles. It showcases how Claude can help create tools - like this very notebook - with little coding skills and no idea of where to start. I am a product manager by profession, and without Claude next to me, I would never have built this.

## Notebook Usage

**Usage – To Fix Your Word Document**

You don't need to know how to code all you need to is set two variables in the Notebook Notebook section "Set your variables." To run the Notebook, you will need an Anthropic Pro account ($20/month) and at least $0.05 credit in your workbench along with an Anthropic API key.

**No Anthropic Pro account?**

No bad problem. Each code block is followed by an example output, allowing you to see how the notebook works without running it yourself. Start by panning the Table of Contents to see where we’re planning and then focus on the explanations between code blocks – you don’t have to read the mode.

## How this Notebook came about

This project fell into my lap tomorrow over a Sunday lunch that I wasn’t at. My sister's friend was wrapping up her research paper - a hefty 40,000-word beast in German. With the deadline looming, she was spending too much time looking for mistakes Word had missed.

So, I wondered to create this Notebook to use Claude to correct Word documents. What makes it different from using the normal Claude chat interface is that the entire document gets corrected, and the corrections are easy to spot as I’ve made them in color.

Did I know how to build something like this? No. But I surmised (correctly) that with Claude's help I could figure it out. And that's really what this project is about - showing just how powerful Claude is when you don’t know how to do something, even when you’re not a coding wizard.

## Claude as a Partner: A Non-Coder's Adventure in AI Development

As I mentioned earlier, I'm a product manager by trade with no heavyweight coding skills to speak of. Out of my depth I sought guidance from an online Slack community. Two strangers, Unmesh and Aaron, generously allowed me to call them. Unmesh validated my stickman concept, while Aaron inspired my "Evaluate the document" section. Both calls gave me the confidence that I knew with persistence I could do this.

I then turned to Claude as my AI co-creator for this Notebook. This marked move from concept to development. Claude became my constant companion, sitting beside me every day in my browser. I created a dedicated [project in my Anthropic account](<https://support.anthropic.com/en/articles/9517075-what-are-projects>) for our chats and gave Claude specific instructions to optimise our collaboration:

Collaborating with Claude was crucial in every aspect of developing this Notebook. Here are handful examples:

* **Problem-solving**: Claude helped me visualize the solution (see the flowchart below)
* **Technical guidance:** Suggested Python libraries to streamline the coding process.
* **API integration:** Provided instructions for connecting to Claude programmatically.
* **Code development:** Wrote about 80% of the code and helped refactor (improve) it.
* **Wrote about 80% of the code and helped refactor (improve) it.**
* **Testing:** Generated test data for prompt testing (see "Test my prompt" section).

Here the picture he drew for me on how we could tackle this Notebook:

[pic question to generate the plan flow]

I learnt an enormous amount, especially when I argued with Claude. I’m a much better coder now too. You probably shouldn’t have thought so, but he explained so much to me. Discussing concepts before diving into syntax made the coding process much easier and faster.

If you are new to AI development, don't be sad. When I started this project, I was starting far from square one, just as you might be. Everything I've learned came from my interactions with Claude. This Notebook is a testament to the power of AI-assisted learning (and code generation).

## Who this Notebook was written for

**Someone with a Word document:** that needs correction beyond what Word provides. In this case, you don’t need to read anything. Just run the Notebook, look at the “Evaluate the document” results, and the final corrected document.

**The Curious Beginner:** If you're feeling like I was at the beginning of this project - knowing it is possible to programmatically use Claude but unsure where to begin - this Notebook is for you.

**For People Good at Coding:** This Notebook showcases Claude's coding capabilities. About 80% of this Notebook were coded by Claude in response to my unsophisticated questions. Yes, there was a lot of back and forth and refactoring, but say what you will, someone who knew very little still built something that works.

**And lastly, for my sister:** Whose definition of AI was Canva and now happy Claude too. I know you won't make it this far, but here's hoping anyway.

# 🟣 Setup

## Python Libraries

Before I dive into actually "doing the work", first I need to install and import the Python libraries this Notebook will use. Libraries are code written by other people which means there's less code for me to write. Many of these libraries were either suggested by Claude or I found them with Google.

**Install all required libraries** if they aren't already:

**Import specific library components** that I'll use in this Notebook.

For example, in `from langchain.text\_splitter import MarkdownTextSplitter` I'm importing the MarkdownTextSplitter class from the text\_splitter submodule of the larger langchain library. Confused? Don’t worry, think of it like this:

* A library is a collection of pre-written code like a toolbox,
* A submodule is a separate file or directory within that library (like a compartment in the toolbox),
* A class is a blueprint for creating objects with specific features and behaviours

# 🟣 Split Word doc into chunks

I initially thought I could simply send your entire Word document to Claude for correction. However, when using Claude programmatically this is not possible. Here's the approach I devised in conversation with Claude this is not possibel. Here's the approach I devized in conversation with Claude

1. Extract the text from your Word document
2. Convert it into markdown format
3. Split the Markdown text into smaller peace (which I'll call "chunks")

Once these steps are complete, I can send each chunk to Claude for processing - in our case, this means getting Claude to correct each text chunk and send it back to me. We gone use this method to insure the whole document gets fixed.

## Extract Word doc

In this first step, I extract the text from your Word document. Because I want to maintain the same headings bullet lists, and paragraphs as your Word document, my code tracks the 'Word style' of each paragraph. For simplicity, I ignored everything else like text found in tables, images, charts, headers and footers.

**The output of the code below** shows a summary of the paragraphs that were extracted from your Word document.

## Split into chunks

Now that I've converted all the extracted text from your Word document into one markdown file, we're onto splitting this file into little chunks . So instead of one big file, I might end up with say, 12 chunks that, if joined back together, would be identical to the markdown file. I do this so I can send these smaller chunks to Claude for correction, one by one.

You might wonder, "Why not just send everything at once?" I think the same. Claude has a limit on how much text it can send back in reply to any prompt. If I sent all the markdown text at once, many contents won’t be left out in Claude's corrected reply.

Small chunks are also quite nifty. They make it easy to compare the "original chunk" with the "processed chunk" I get back from Claude. For instance, I can quickly spot if the semantic meaning of the corrected chunk changed compared to the original chunk. It's much easier to notice oddities with little chunks than one enormous chunk. Plus, if something goes wrong, I'd rather have a small chunk fail than the entire document.

In the output of the code below you can see a summary of the chunks your Word document got split into. The number of chunks is determined by the target chunk size I set. More on that below. But feel free to skip it entirely.

Why I chunked with characters (not tokens)

I chose to split the text into chunks based on character count rather than tokens. For instance, "split the document into chunks each about 1,000 characters long." It's a bit of a rabbit hole, but if you're curious as to why I chose characters (instead of tokens), here's the exploration:

First, let's talk about "tokens." They're just a way to measure text size, like kilograms measure a person's weight. But here's the twist: while a kilogram is the same everywhere, tokens aren't. Each large language model has its own way of counting text size using their own "tokeniser". So "87 Anthropic Claude tokens" isn't the same amount of text as "87 OpenAI GPT tokens". Language models don't measure the size of text by counting words or characters like we do.

Given that Claude is limited to replying with no more than 4,096 "Anthropic tokens", it would obviously make sense to split text by counting tokens. So why did I use characters instead?

I got a bit lazy. My text is in Markdown format, and I didn't want to break apart Markdown elements accidentally when splitting the text into chunks. I found a library that splits Markdown text while respecting its structure. The problem is that this library only allows me to specify target chunk size in characters (not tokens). Still, I thought it better to do a workaround than to write my own code to split markdown text. I thought doing rough math estimates were easier.

**How I determined my target chunk size (in characters)**

To determine my ideal chunk size (in characters),first I got my bearings on counting tokens, characters, and words on some random text using the Anthropic Tokeniser and Python.



Then I did some math:

1. Claude can return a maximum of 4,096 tokens per chunk
2. My equation: (prompt size) + (chunk size) + (wiggle room for corrections) <= 4,096 tokens
3. Anthropic’s tokeniser measured my prompt (without a chunk attached) at 405 tokens.
4. I decided 500 tokens for correction wiggle room would be enough.
5. So: (405 tokens) + (chunk size) + (500 tokens) <= 4,096 tokens
6. This means that my chunk size must be no more than 3,191 tokens

And now using the data above to approximate:

3,191 tokens x 5.89 characters = 18,795 characters

But hold on, 18,795 characters in English is about 2,565 words which is 5 pages in Microsoft Word. I want smaller chunks to pinpoint subtle changes more easily. In my prompt I instruct Claude to “make corrections but retain original meaning.” Having smaller chunks to compare means I can plan changes in meaning with more sensitivity.

I chased 4,000 characters as my target chunk size because this is about 545 words (just less than one page in Word) and about (4,000/5.89) 679 Anthropic tokens. Remember my wiggle of 500 tokens? This is about 400 words which is more than enough as I’m only sending Claude 545 words for correction any wow.

My numbers are rough approximations using the sample analysis data above. But with a target chunk size of 4,000 characters (or 679 tokens) there is 2,512 tokens of room left anyhow:

So: (405 tokens) + (chunk size: 679 tokens) + (500 tokens) <= 4,096 tokens

The only downside I see is a little more cost. Costs for API usage is calculated based on the number of tokens used, not per prompt. Because I’m sending more prompts than I need to, it just means I’m sending 405 (i.e., the prompt measured without the chunk) tokens more times than I have to. That’s okay.

**In the output of the code below,** you’ll see a summary of how your Word document got chunked. You’ll also see that my math isn’t all that bad.

# 🟣 Prompt Claude to correct chunks

## Show Prompt with example chunk

## Access Claude via API

If there was seven part that scared me the most about attempting this Notebook, it's this part. I've always struggled with reading and understanding API documentation API stands for "Application Programming Interface" -- in our case the "application" is Claude, and "programming interface" means I'm using my Python program (ie this Notebook) to call Claude.

To get around my nerves, I told Claude that he is an expert on the latest Anthropic API. I also gave him the latest Python API documentation I downloaded from Anthropic's website and told him to educate himself deeply. I did this because the Claude model I'm using was last updated in June, and there has been an API update since then that Claude wouldn't be aware of. Finally, I asked him to write the code to connect to the Claude model. I pasted his code below and hit run. That's it. Claude replied and I did a happy dance to be sure.

## Process Chunks

Let's summarize where we are so far. We've extracted the text from your Word document, we've converted it into a markdown file, we've split that file into chunks of markdown text, and we've established we can connect to Claude using the Anthropic API.

In this step I send each chunk one by one, embedded into the prompt, to Claude. The prompt instructs Claude to make corrections to the text. For each chunk, Claude then processes it and sends the corrected chunk back to me. I collect these chunks back in order because I'm going to join them all altogether again into one big document.

The code is quite long but only because I went to town on error handling and friendly error messages. I did this because I can never understand error messages, so at least now mine are friendly. I also put in a fancy progress bar to keep you entertained while the chunks are processing.

# 🟣 Show word doc corrections prettily

At this stage, I've sent all the text chunks extracted from your Word document to Claude. He's processed each as per the instructions in the prompt and sent them back to me. The result of the two steps below is to enable you to see your entire Word document along with colourful corrections in your browser.

## Reassemble processed chunks

This step is straightforward, just join the processed chunks back together. Since I collected the processed chunks from Claude in the same order I sent them, they fit right back into their original spots, following the flow of your Word document. The result? One big markdown file containing all the corrected text. It's essentially your original document, but with Claude's corrections neatly incorporated.

## Create a pretty HTML file

Now that I have a markdown folder that contains all the done chunks, we’re ready for this step. I convert this file into an HTML file so it looks prettier for you and it’s easier to spot the corrections. All corrections are both bold and, now thanks to HTML, can also be in colour. The original scaffolding of your Word document has also been maintained. In terms of output, once the below code has run, we’re done. Enjoy your corrections Word likely didn’t pick up. They’re all easily visible in the HTML file that you can open in your browser.

But my work isn’t done. I’m onto testing: the output file, my prompt itself, and my code.

# 🟤 Check corrected document

At this point, we have two very useful collections: the first contains all the original chunks before we sent them to Claude. The second contains all the corrected chunks that were processed and corrected by Claude. I tested my prompt, and the results were quite stable in retaining original meaning. However, I still wanted an automated way to know this for every Word document corrected.

So, I set up three different ways to evaluate the corrected text against your original text. These provide automated confidence that Claude has enhanced your writing without changing it's core message, structure, or content.

Here's what I'm checking:

1. Document structure: Are the headings and paragraphs still in the right place?
2. Document content: Word count – has the overall length changed dramatically?
3. Document content: Semantic meaning – does the corrected text still mean the same as the original?

Each section below will go into more detail.

"Michelle you are making overkill," I hear you saying. I quite disagree, and here's why:

Imagine I decide to switch to the free Meta Llama 3.1 model instead of Claude Sonnet. How would I know if it's performing as well as Claude did? What if I have 20 large Word documents to correct? It wouldn't be practical to manually check each one.

Imagine I decide to switch to the free Meta Llama 3.1 model instead of Claude Sonnet.

Moreover, even if I stick with Claude, how can I be sure he'll perform equally well in Italian as he does in English? And just because I've thoroughly tested one document, does that guarantee the same level of performance for all others?

Manually testing for all these scenarios would be incredibly time-consuming. That's why these automated tests, checks, or evaluations (whatever you want to call them) are so important to me. They save a tremendous amount of time and provide consistent, reliable results for every Word document I correct, regardless of its size or language. With very little effort, I can now have confidence in the quality of corrections across a wide range of documents and potential future changes.

## Document structure Preservation

This checks whether the overall structure of your Word document, like headings and bullet lists, remains intact in the corrected document. It's like making sure the skeleton haven't been rearranged. Keep in mind that Claude might combines some paragraphs while he's doing correction. For example, if you accidentally started a new paragraph mid-sentence in your Word document, Claude would probably fix that. So don't judge an imperfect match too harshly. It's just an indication that something have changed. Overall, I don't expect that too much will change. This is what I am checking.

## Document Content Preservation: word count

Here, I do a quick comparison of word counts. If there's a big difference between the original and processed word count from Claude, it might mean the content has change too much. It's a simple but effective first check that helps us spot any major unexpected changes in content. If you do see something unexpected, then it's a flag to go and look. Overall, here too, I don't expect word count to change by a lot, and this is what I am checking.

## Document Content Preservation: meaning

2 This is the most sophisticated check of the three, diving into the actual semantic meaning of the text. There are many ways to measure text similarity. It sent me down another rabbit hole with many conflicting conversations with quite a few people. I've chosen to use the sentence-transformers library, which loads a language model that I selected to understand and compare text meaning.

# 🟤 test my prompt

Creating the perfect instructions (or "prompt") for Claude is a mix of creativity and precision. It took me quite a bit of trial and error of refining the prompt to get Claude perform like I wanted him to. As I was tweaking my prompt, I kept manually retesting the same things. That is why I wrote these tests -- to automate my manual tests so I could tweak faster with more confidence. They send example chunks to Claude to ensure that no matter the tweak, he is still doing the very specific things I care about, like correcting inappropriate word choice or bolding a correction using markdown format.

For more encompassing testing, of both Claude prompt and code together, see section 'Evaluate the output file.' In these prompt tests, I isolate the testing to only the prompt. None of my other code is involved in any sort of way. That was important.

You'll see in the output below that there are two failing tests. Try as I might, I just couldn't get Claude to consistently detect British English from American English and do corrections in the detected variant. As someone who prefers the Queen's English, I spent a lot of time trying to craft the prompt so Claude would get this right every time. Finally, I gave up and removed all associated guidelines. American English it is.