Spreadsheets with superpowers

Dataharvest 2025

Pre-requisites

- You need access to Google Sheets
  - If you don’t have a Google/Gmail account already, **you will need to create one**
- If you have your own Anthropic or OpenAI API keys, please have them handy
  - If not, I’ll provide one for the duration of the workshop
- If you have an example of a spreadsheet you’ve worked on where you think an LLM function could be useful, please bring it along so you can try it out!

What will we cover?

1. How to use Google’s Apps Script to create a custom FUNCTION() in Google Sheets
2. How to return single/multiple values from your FUNCTION()
3. How to fetch something from the internet (e.g. from an API) in your FUNCTION()
4. How to call the Anthropic API in a custom function called CLAUDE()
5. Some good use cases for an LLM-powered function like CLAUDE()
6. *[maybe, if time]* CLAUDE() => LLM(), calling OpenAI API as well as Anthropic

What will we **not** cover?

- How to get AI to generate your Apps Script code for you
    - We recommend you do this in future—but it's easy
    - If you do this *first*, you won't know how anything works, which will make it hard to do more complicated stuff
    - We want to build the habit of doing the following before using AI:
        - Finding the correct documentation
        - Building a simple toy example from first principles
- All the potential use cases or pitfalls of having an LLM() function
- How to do this outside of a Google Sheet
- How to make your function available in multiple sheets
    - This would involve publishing your sheet as an Editor Add-on

Wait, aren't there already LLM-powered Sheets functions?

- Google provides an AI() function which uses Gemini
  - But there are lots of caveats around whether it will be available to you right now
- Anthropic provide Claude For Sheets
  - But mixed reviews
- There's also GPT for Sheets™ and Docs™ from a third-party (Talarian)
  - But paid features
- Knowing how to build one from scratch will give you flexibility and freedom

Step one: Get your own copy of the Google Sheet

The sample data we'll test drive our function with is from Political Advertisements from Facebook on Kaggle.

1. Open up this Google Sheet: [Dataharvest] US Political Ads on Facebook
2. File > Make a copy

Step two: Write a simple ECHO() function

Documentation: Custom Functions in Google Sheets | Apps Script

Go to “Extensions > Apps Script” in the menu

Let’s code an ECHO() function that:

1. Returns a single cell
2. Returns multiple columns
3. Returns multiple rows and columns

Code: SPOILER ALERT!!!

https://gist.github.com/joelochlann/6fd57a1bcc108ee7d877c29b7fc5987e

Step three: Write a simple COIN_FLIP() function

We’ll use URL Fetch Service to call an API

1. Try calling a broken URL to see what happens
2. Call https://api.toys/api/coin_flip and return response directly
3. Parse JSON and return just the result of the coin flip

Code:

https://gist.github.com/joelochlann/11ce3281c1e40c62cb5cbca82f1c2d92

Step four: get an API key for Anthropic

Anthropic are a rival to OpenAI. Their equivalent of ChatGPT is called Claude, and is powered by models like Claude Sonnet 3.7.

To use these models directly, instead of via a chatbot UI, we’re going to use the Anthropic API.

You can sign up for Anthropic API access here: https://console.anthropic.com/

Once you add some money to the account, you can issue an API key which will then be billed by usage. This usage is the number of tokens (roughly, words) that you send to Claude (input), and that Claude responds with (output).

For many use cases, you will spend very little as input is charged at **$3 per million** tokens and output at **$15 per million** tokens.

Step five: write a CLAUDE() function

The official documentation is **even more important** now in the age of AI. When your AI-generated code doesn't work, the answer **is usually in the documentation**.

1. Translate the "hello world" cURL example from the Anthropic docs into an Apps Script function, CLAUDE_HELLO_WORLD()
2. Initially return `response.getContentText()`. The JSON will look a bit ugly, so let's pretty print (`pbpaste | jq` if you're handy in the terminal, else JSON Pretty Print)
3. Now that we know the structure, return just the model's response
4. Drag down our new formula
5. Increase temperature to show greater variation
6. Make a new function, CLAUDE(), which accepts a prompt parameter
7. Try prompting with CLAUDE("What is the capital of France?"), then with list of questions, then try some it can't answer well because of training cut-off (recent ev SPOILER ALERT!!! s)

Code: CLAUDE_HELLO_WORLD(), CLAUDE()

Step six: using our CLAUDE() function

1. Add a row above the headers to contain the fixed portion of our prompt
2. Add a formula to combine the prompt with the ad text. **Formula** here SPOILER ALERT!!!
3. Drag down to see how prompt will look
4. Apply CLAUDE() to our prompt, and drag down
5. Check out the results using filters!

⚠️ **All these API calls will re-run when you reload the sheet later**

💡Copy calculated results (values only) and remove formulas to “cache” results

Finished product: SPOILER ALERT!!!

[Dataharvest] Political ads about immigration

Going further: calling different LLMs from one function

1. CLAUDE() => LLM()
   - Accept model as a parameter
   - Add OpenAI as an alternative API (https://platform.openai.com/docs/api-reference/chat/create)
   - Then you can do
     - =LLM("gpt-4.1","say hi")
     - =LLM("claude-3-7-sonnet-20250219","say hi")

SPOILER ALERT!!!

Code: https://gist.github.com/joelochlann/d8fd444aedeb4b31ab3f564f23ff2a72

Limitations and caveats

- The function is tied to this sheet
- All these API calls will re-run when you reload the sheet later (costing you money and potentially giving different results!). You may want to copy/paste prior results

Limitations and caveats: **LLMs are not always reliable**

- They have no memory beyond their training data so will not have knowledge of recent events
- They are not fully deterministic, **even with temperature at 0**
- Their knowledge acquired through training is “fuzzy” (probabilistic) and dependent on the quality of information seen in training
- Using them for natural language processing (NLP) tasks like classification (as we did here) or extraction is likely to be more reliable than knowledge-based tasks, **but will still not be 100% accurate**

Best practices for using an LLM on your data

- Test first on a subsample and calculate performance. (The best metric will be task-dependent, but for classification tasks this is a good place to start)
- If you have the expertise, calculate a confidence interval based on the number of samples. This will tell you how confident you can be that it will have the same accuracy on your full dataset

ENDS