Optimizing GPT Prompts for Data Science

datacamp

Andrea Valenzuela

on behalf of

ForCode'Sake.

Welcome

to the first step of

YOUR data-driven journey

with

ForCode'Sake.

Not an ordinary company.

But a **passionate collaboration** of two **college friends** that...

had so many crazy ideas and a shared **passion for Data Science.**

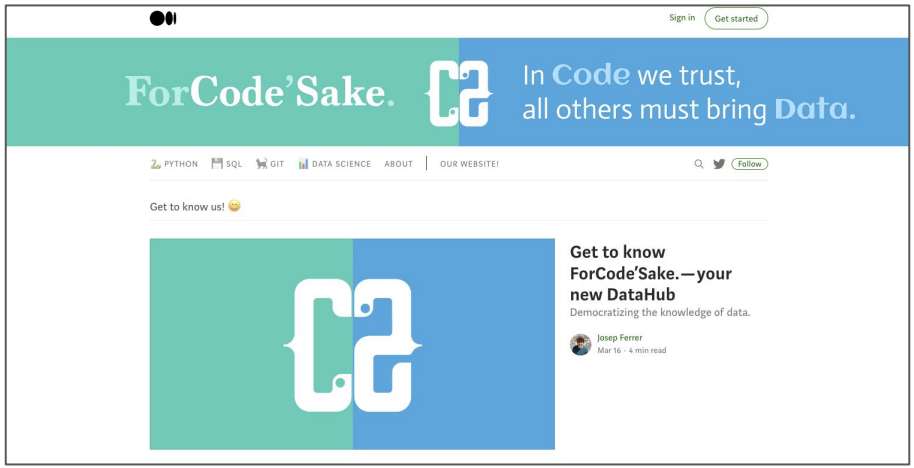
The Team

Josep Ferrer
Analytics Engineer
Focused on social and human mobility data.
Data Science technical writer on Medium and collaborator of KDnuggets.

Andrea Valenzuela
DevOps in Physics
Working at CERN as a computer engineer.
Data Science technical writer on Medium.

About ForCode'Sake

Started as a consultancy duo, we evolved into a Medium publication where we share curated stories on Data Science and Programming ✨



You can go check our publication [here!](#) 🤓

The Webinar - Optimizing GPT Prompts for Data Science

This webinar has three main goals:

- 🎯Learn the principles of good prompting engineering when using ChatGPT and the GPT API
- 🎯Learn how to standardize and test the quality of your prompts at scale
- 🎯Learn how to moderate AI responses to ensure quality

⚠ To follow the webinar, you need to have an active OpenAI account with access to the API.

No idea where to start? You can [follow this tutorial to set up and activate your OpenAI account](#).

1. Principles of Good Prompting

The first important message about prompting is seeing it as **an iterative process.**

→ Best prompts are created after several iterations of testing and improvement.

We will review 5 different ways to enhance our prompt when doing an iterative process:

1. **Give details.** By being clear and giving ddetials one can guide the model towards the desired output.
2. **Use delimiters.** Delimiters allow one to distinguish between different parts of a prompt and avoid prompt injection.
3. **Few-shot prompting.** Train the model with examples to enhance its capacities in specific domains.
4. **Specify the intermediate steps of a task.** Always define a chain of relevant reasonings before the model provides its final answer.
5. **Bear the tokenizer in mind.** It is important to always consider that models do not “think” using words, they use tokens.

1. Principles of Good Prompting

Related Resources:

- Stop doing this on ChatGPT and get ahead of the 99% of its users. Easy to understand tutorial on how to craft good prompts.
- Unleashing the ChatGPT Tokenizer. Not familiar with the concept of token? Break it down together with Andrea!
- Prompt Engineering Course by OpenAI. Have you missed the Deeplearning.ai course? Here you will find a power summary.

2. Prompts On-Scale

LLM models work extremely well when answering user requests. Users usually take an active role to craft good prompts and check the output so everything works fine.

However, **what happens when we want to scale this model?** When users prompts become large, we need automatic ways to test and control the quality of this output.

This is why there are two main best practices that we can consider when writing our prompts:

- Ask for a **structured output**, in order to further standardize your tests.
- **Control outlier responses** from the model. That is, restricting the model *freedom* when facing unseen or unexpected inputs.

Related Resources:

- The Future of Sample Data Generation — Unleashing the Potential of ChatGPT. Generate sample data using the GPT model while learning how to define structured output.

3. AI Moderation

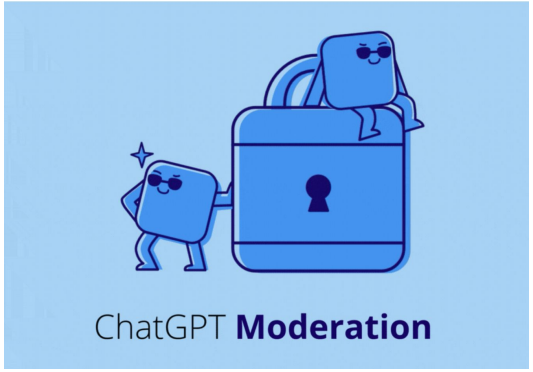
LLM accuracy - as any machine learning algorithm - is strongly affected by the training data. And as humans, **the data we generate is always biased.**

This is why it is crucial to recognize the significance of controlling and moderating user input and model output when building applications that use LLMs underneath.

There are two main ways:

📥 **User input control** refers to the implementation of mechanisms and techniques to monitor, filter, and manage the content provided by users when engaging with powered LLM applications.

📤 **Output model control** refers to the implementation of measures and methodologies that enable monitoring and filtering of the responses generated by the model in its interactions with users.



3. AI Moderation

In this tutorial, we propose two **Moderation techniques**:

1. The usage of GPT models to evaluate the output given by another GPT model.

2. The usage of the OpenAI Moderation API to ensure correctness of the user inputs and model outputs.

Related Resources:

- ChatGPT Moderation API: Input/Output Control. Further understand the ChatGPT Moderation API.
- Building Context-Aware Chatbots. Learn how to create function chatbots with context-aware memory.

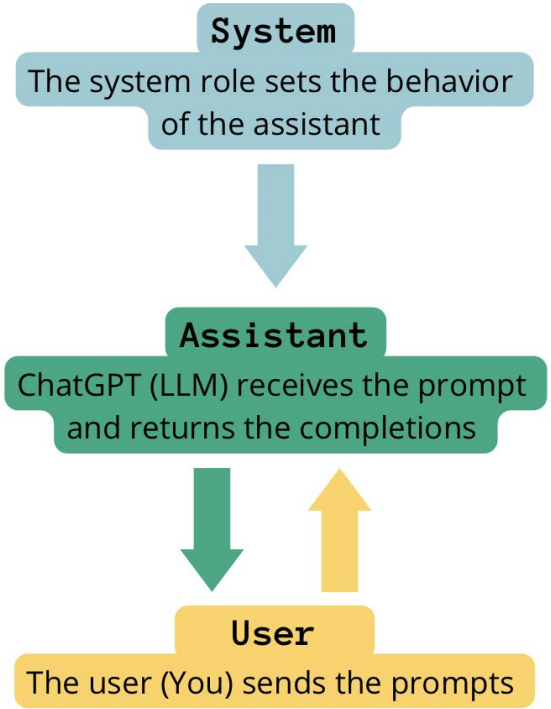
TUTORIAL CONCEPTS

Workspace link here!

SYSTEM MESSAGES

Let's consider that we are building an application where an external user interacts with the GPT model.

- It is important to define the **high-level behavior** of the model.
- The high-level behavior of the model refers to **its primary generic purpose within the LLM application**.
    - We can fine-tune the generic GPT behavior with the so-called **system messages**.
- The system message can be presented to the model in a transparent way to the user.



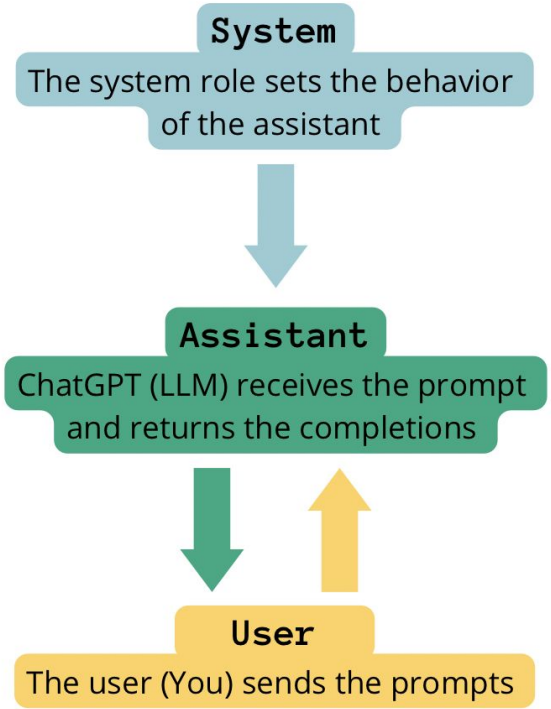
SYSTEM MESSAGES

In the Notebooks, our prompts will have two parts:

```
prompt = f"""
You are a data science expert.
Your task is to define any term given by the user.
Output only the given term and a short definition.

{user_text}
"""
```

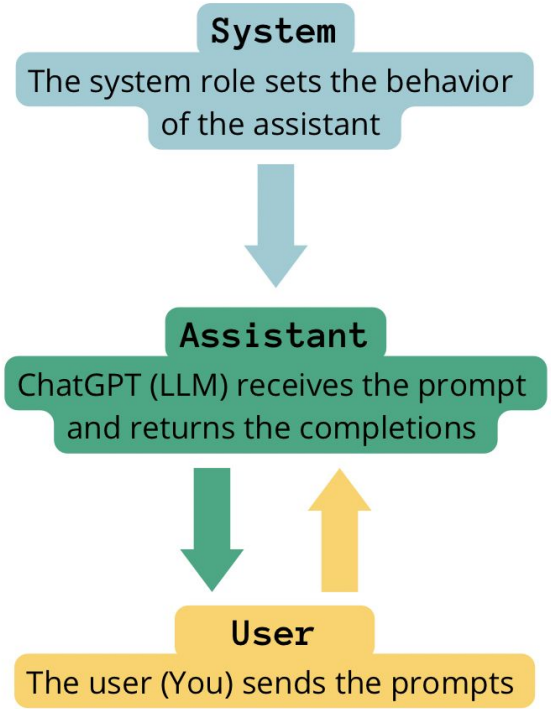
```
user_text = """
Can you give me a definition of the Data Science term?
"""
```



GPT CONVERSATION HISTORY

```python
def chatgpt_call(prompt, model="gpt-3.5-turbo"):
    response = openai.ChatCompletion.create(
        model=model,
        messages=[{"role": "user", "content": prompt}]
    )
    return response.choices[0].message["content"]
```

```python
messages =  [
{'role':'system', 'content':'You are friendly chatbot.'},
{'role':'user', 'content':'Hi, my name is Andrea'},
{'role':'assistant', 'content': "Nice to meet you, Andrea!
{'role':'user', 'content':'Do you remember my name?'}  ]
```



GPT CONVERSATION HISTORY

```
messages = [{'role': 'system', 'content': chatgpt_system_message}]

def chatgpt_call(prompt, model="gpt-3.5-turbo"):
    messages.append({'role': 'user', 'content': prompt})
    response = openai.ChatCompletion.create(
        model=model,
        messages=messages
    )
    response_text = response.choices[0].message["content"]
    messages.append({'role': 'assistant', 'content': response_text})
    return response_text
```

GPT memories can be built in an optimized way. Check it out here!

AI MODERATION

CUSTOMER SERVICE AGENT

System message:
"You are a customer service agent [...]
This is the catalog of our products [...]"

User input:
"Which TVs do you offer?"

Agent (GPT model):
"According to our catalog [...]"

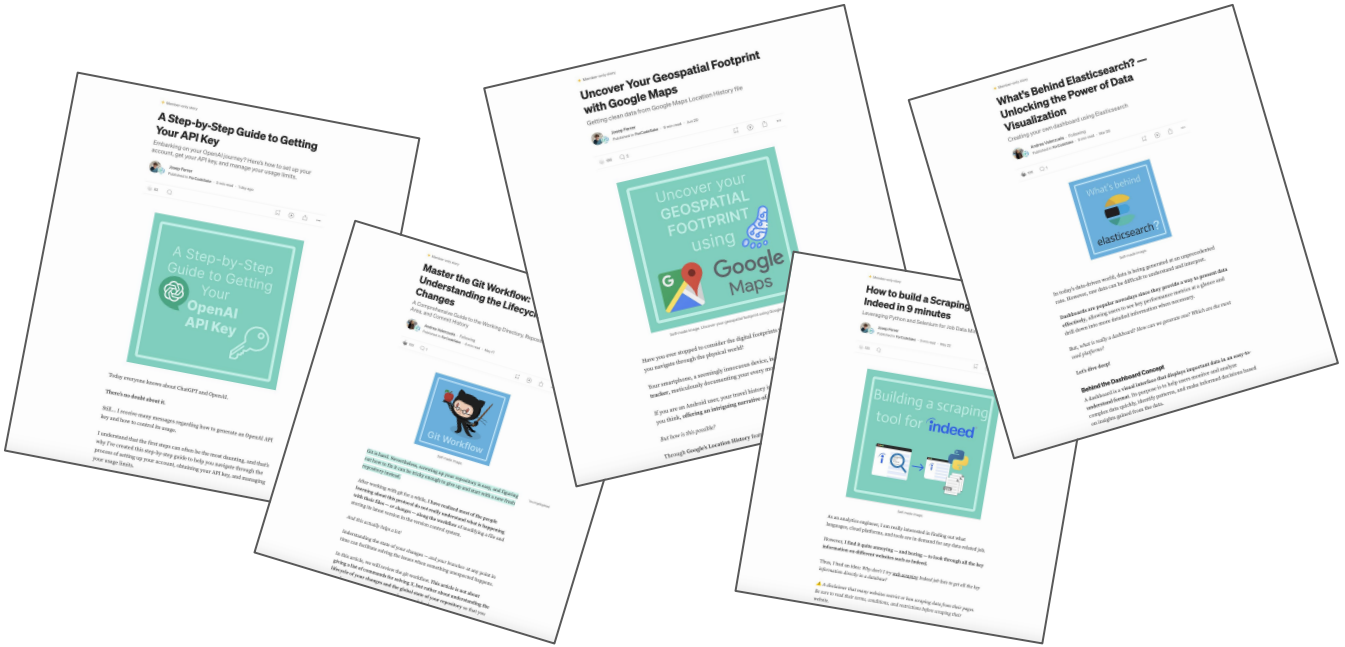
QUALITY (QA) AGENT

System message:
"Your task is to evaluate the customer service agent response given a user question [...]"

User input:
"This is the catalog of our products [...]
This was the user question [...]
This was the model answer [...]"

Agent (GPT model):
"The answers is correct/sufficient [...]"

More Material of ForCode'Sake.



Did you enjoy the webinar and want to be updated with our latest tutorials? ✨

[Just follow our newsletter!]

In Code we trust,
all others must bring Data.

-by ForCode'Sake.

Thank you! Q&A time!

You can also forward your questions to: *forcodesake.hello@gmail.com*