**Solution #1**

Me: “I'm a student doing a Python project in a DH course. Act as a good tutor and give

me advice without giving me the full answers.”

ChatGPT said:

Love that approach — let’s do this like a proper coding mentorship session. 👌

Tell me what your project’s about, where you’re stuck, or what you need advice on. I’ll nudge you in the right direction, help you think through it, maybe drop a little hint or two, but you’ll be steering the ship.

What’s on your screen right now?

You said:

I have encountered a problem during my DH project. I have a whole bunch of files to deal with. I have to clean the names of the places in files. Please help me how can I approach it using the normalize function.

ChatGPT said:

Awesome — you’re tackling a classic Digital Humanities (DH) data cleaning problem. Handling messy place names is totally a thing in DH projects. Let’s break down how you might approach this without giving away your code.

📌 About normalize

If you're using a custom normalize() function (or writing one), it generally means:

Stripping extra whitespace

Standardizing case (like .lower() or .title())

Removing or replacing unwanted characters

Dealing with Unicode issues (like accented letters)

📌 Possible Approach (in pseudocode form):

python

Copy

Edit

def normalize(place\_name):

# 1. Remove leading/trailing whitespace

# 2. Convert to lower case (or title case depending on what you need)

# 3. Remove unwanted symbols or numbers

# 4. Maybe handle accented characters if necessary

return clean\_name

📌 Pro tips:

Python’s str methods like .strip(), .lower(), .replace(), .title() will be your friends here.

For removing accents: look into unicodedata.normalize() — not mandatory, but cool if your data has accented names.

Always test your function on a few sample names before applying it to your entire dataset.

If the names are in multiple files — consider reading them all into a list or DataFrame first, cleaning them there, and then writing back.

📌 Questions for you:

Are your place names simple strings, or are they embedded in a larger structure (like a CSV row or JSON object)?

Do you have examples of before and after you’re aiming for? That can help shape your normalize() design.

Would you like to show me a couple of sample place names or your rough code for normalize()? I’ll happily review and give you pointers 👌

You said:

Oky but I have certain names such as

'britain': 'United Kingdom', which are same names, but they are being treated as two different places in my code. How should I approach it?

ChatGPT said:

Excellent — now you’re hitting one of the classic DH problems: standardizing place names where different labels mean the same thing. “Beruit” vs “Beirut”, “Britain” vs “United Kingdom” — classic data cleaning pain.

Let’s strategize without me handing over full code:

📌 Two-part Strategy

Part 1: Basic Normalization

Your existing normalize() function should handle cleaning like:

Lowercasing

Removing extra spaces

Fixing punctuation

But this alone won’t fix cases like 'britain' vs 'United Kingdom' because those are semantically different, even if spelled cleanly.

Part 2: Mapping Synonyms to Standard Names

You’ll want to build a mapping dictionary for cases like this. Something like:

python

Copy

Edit

place\_corrections = {

'beruit': 'Beirut',

'britain': 'United Kingdom',

'u.k.': 'United Kingdom',

'united kingdom': 'United Kingdom'

}

Then, in your normalize() or after normalizing, check if the cleaned name is in this dictionary and replace it with the corrected one.

📌 Possible Workflow (in pseudocode)

python

Copy

Edit

def normalize(place\_name):

# clean name: strip, lower, etc.

cleaned = place\_name.strip().lower()

# check if cleaned name is in corrections

if cleaned in place\_corrections:

return place\_corrections[cleaned]

else:

return cleaned.title() # or however you prefer to format your final names

📌 Advice:

Build your place\_corrections dictionary iteratively. Start with obvious issues you spot, then expand as you find new variations.

Log or print unique place names after initial normalization so you can see what odd variants you’re dealing with.

Don’t be afraid to add multiple synonyms per place.