“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.” i have to accomplish the below task Improve the recall of place names: In class, we used only the first column (“asciiname”) of the gazetteer to identify a place in a text. We noticed that this did not catch all mentions of the place in an article (because of spelling differences) - in computer science language: the recall was low. Adapt the script so that it becomes better at identifying place names from the gazetteer in the text. Strategies you could use: ○ Instead of using only the asciiname column to identify a place in the text, make your script build (for each row in the gazetteer) a regular expression in your script that matches either the asciiname column or any name in the other columns that contain alternative names for the place. E.g., for “Khan Younis”, your script could build a regex like this: “Khan Younis|Khan Younis|Khan Younis|Khan Younis camp|KhanYounis|Mukhayyam Khan Yunis” ○ OR, build a regex pattern based on the asciiname for each row, by replacing each character in that asciiname that could be spelled differently by the alternative options for that character. E.g., a regular expression that matches all spellings of the asciiname “Khan Younis” could be: “[KhḪ][aā]n Y[ouū]\*[ouū]\*n[ie]s”. You can build this regex pattern by replacing: ■ “Kh” in “Khan Younis” with “[KhḪ]”, ■ “a” with “[aā]”, ■ “o” or “u” with “[ouū]\*”, and ■ “i” with “[ie]”. By comparing the asciiname column with the alternative names column, you should be able to come up with other characters you can replace with a regex in the asciiname to match more different spellings of the name. NB: you can use the website https://regex101.com/ to check how well your regex works; ○ OR, feel free to come up with your own clever way to improve recall!

**ChatGPT said:**

Awesome — love seeing people tackle this kind of text-matching problem in DH (Digital Humanities)! Let’s break this down tutor-style, and I’ll steer you without spoiling the whole thing.

**📌 What’s really happening here:**

Your current approach is **low recall** because it relies on a single version of a place name (the asciiname). But in real-world texts, names get spelled differently, might be written with/without diacritics, or might even have nicknames/variants.

**Your job:** improve recall by matching more variations.

**✅ Here’s how you might approach it (conceptual steps):**

1. **Understand your data**
   * Look at your gazetteer file (probably a CSV or TSV).
   * Note the columns — typically there’ll be asciiname and maybe alternatenames or other columns listing name variants.
   * What kind of variants do you see? Are they separated by commas? Are there diacritics?
2. **Decide your strategy**
   * The exercise suggests two possible strategies:
     1. **Match against all known variants** — use both asciiname and any alternatenames in your regex pattern.
     2. **Create regex character groups** to account for spelling differences.

**Good advice:** Start with option 1 because it’s conceptually easier, then if time permits, experiment with option 2.

1. **For Option 1: Match known variants**
   * For each row:
     1. Grab the asciiname
     2. Split the alternatenames (if it’s a comma-separated string)
     3. Combine them into one regex pattern using the | (OR) operator.
     4. Escape special regex characters like (, ., + etc.
   * Example:  
     If asciiname is Khan Younis and alternatenames are Khan Younis Camp, KhanYounis, Mukhayyam Khan Yunis  
     → regex: Khan Younis|Khan Younis Camp|KhanYounis|Mukhayyam Khan Yunis
2. **For Option 2: Build character-group regexes**
   * Look for common substitutions (like a ↔ ā, i ↔ ie)
   * Replace those in the asciiname using regex brackets
   * Example: Khan Younis → [KḪ]h[aā]n Y[ouū]\*[ouū]\*n[ie]s
   * This is trickier but works when you *don’t have alternatenames*.
3. **Implement a lookup loop**
   * Read your gazetteer into a data structure (list of dicts or a pandas DataFrame if allowed)
   * Iterate over each place
     1. Build its regex
     2. Use re.findall() or re.finditer() on your text to see if there’s a match
   * Keep track of matches you find
4. **Test your regexes**
   * Use <https://regex101.com/>
   * Paste your pattern and a sample text to see if it catches what you expect.

**⚡ Tutor tips:**

* **Clean your text and names**: lowercase both or remove punctuation for consistent matching.
* **Escape regex special characters** in place names before combining them.
* **Use raw strings for regexes in Python**: r'pattern'
* **Modularize your code** — write a function like build\_regex(place\_row) so you can swap strategies easily.
* **Debug small pieces first** — test regex patterns for one place before scaling up.

Would you like me to help you draft the *function structure* for one of these approaches (without the full code)? I can sketch that out for you too.

**You said:**

#importing regular expressions to find text patterns import re #importing os to enable interaction with the file system import os #for handling tabular data and exporting tsv import pandas as pd #function that writes a list of data rows into a tsv file using pandas def write\_tsv(rows, column\_list, path): #convert the list of rows into pandas DataFrame df = pd.DataFrame(rows, columns=column\_list) # write the dataframe to tsv: df.to\_csv(path, sep="\t", index=False) # Defining the folder where the articles are present # these articles are text files to search for place names folder = r"C:\Users\Dell\Downloads\FASDH25-portfolio2\articles" # define the path and load the gazetteer from the tsv file, having place names and alternate names path = r"C:\Users\Dell\Downloads\FASDH25-portfolio2\gazetteers\geonames\_gaza\_selection.tsv" #open and read the file with open(path, encoding="utf-8") as file: data = file.read() # Create an empty dictionary to store regex patterns for each place and how many times they appear patterns = {} # Split the gazetteer data rows = data.split("\n") # Skip the header row and process each remaining row one by one for row in rows[1:]: # Split the current row into its columns (TSV = tab-separated values) columns = row.split("\t") # If a row doesn't have at least 6 columns, skip it (incomplete data) if len(columns) < 6: continue # The first column contains the main place name (asciiname) asciiname = columns[0] # Create a list to store this place's name and its alternate versions name\_variants = [asciiname] # The 6th column (index 5) contains alternate names, separated by commas alternate\_names = columns[5] # Check if there are any alternate names listed if alternate\_names: # Split the alternate names by comma into a list alternate\_list = alternate\_names.split(",") # Go through each alternate name for alternate in alternate\_list: # Remove any extra spaces around the alternate name alternate = alternate # If the cleaned alternate name isn't empty, add it to the list of variants if alternate: name\_variants.append(alternate) # Use re.escape() to safely handle special characters in each name variant for regex name\_variants = [reescape(name) for name in name\_variants] # Create a regex pattern that matches any of the name variants # \b makes sure we match whole words (e.g., 'Gaza' won't match 'Magazine') regex\_pattern = r"\b(" + "|".join(name\_variants) + r")\b" # Save this pattern and start the match count at 0 in the patterns dictionary patterns[asciiname] = {"pattern": regex\_pattern, "count": 0} this is my code after running im getting an error: ==================== RESTART: C:/Users/Dell/Pictures/dwf.py ==================== Traceback (most recent call last): File "C:/Users/Dell/Pictures/dwf.py", line 72, in <module> name\_variants = [reescape(name) for name in name\_variants] NameError: name 'reescape' is not defined Act as a good tutor and give me advice without giving me the full answers

**ChatGPT said:**

Excellent work so far — you’re really close!

Let’s read the error carefully together:

pgsql

CopyEdit

NameError: name 'reescape' is not defined

**🔍 What’s happening?**

Your code is trying to call a function called reescape(), but Python doesn't know what that is — because there’s no built-in or imported function with that name.

**What you meant to use is:**  
👉 re.escape()

This comes from the re module you imported at the start:

python

CopyEdit

import re

**Tip as your tutor:** Whenever you see an error like this, check whether the function you're calling is one you defined, or one from a module — and whether you spelled its name exactly right.

In this case, it’s a simple typo:  
✅ re.escape() not ❌ reescape()

**📝 Small extra tip**

While you’re fixing that, also consider stripping whitespace from the alternate names (since you did .strip() in your comment but not in your code). Tiny improvements like that make a regex more reliable.

Example:

python

CopyEdit

alternate = alternate.strip()

Would you like me to help you sketch a little test to check if your regex patterns work on a sample text before plugging them into the full article search? Great habit to build 👌

4o

**You said:**

#importing regular expressions to find text patterns import re #importing os to enable interaction with the file system import os #for handling tabular data and exporting tsv import pandas as pd #function that writes a list of data rows into a tsv file using pandas def write\_tsv(rows, column\_list, path): #convert the list of rows into pandas DataFrame df = pd.DataFrame(rows, columns=column\_list) # write the dataframe to tsv: df.to\_csv(path, sep="\t", index=False) # Defining the folder where the articles are present # these articles are text files to search for place names folder = r"C:\Users\Dell\Downloads\FASDH25-portfolio2\articles" # define the path and load the gazetteer from the tsv file, having place names and alternate names path = r"C:\Users\Dell\Downloads\FASDH25-portfolio2\gazetteers\geonames\_gaza\_selection.tsv" #open and read the file with open(path, encoding="utf-8") as file: data = file.read() # Create an empty dictionary to store regex patterns for each place and how many times they appear patterns = {} # Split the gazetteer data rows = data.split("\n") # Skip the header row and process each remaining row one by one for row in rows[1:]: # Split the current row into its columns (TSV = tab-separated values) columns = row.split("\t") # If a row doesn't have at least 6 columns, skip it (incomplete data) if len(columns) < 6: continue # The first column contains the main place name (asciiname) asciiname = columns[0] # Create a list to store this place's name and its alternate versions name\_variants = [asciiname] # The 6th column (index 5) contains alternate names, separated by commas alternate\_names = columns[5] # Check if there are any alternate names listed if alternate\_names: # Split the alternate names by comma into a list alternate\_list = alternate\_names.split(",") # Go through each alternate name for alternate in alternate\_list: # Remove any extra spaces around the alternate name alternate = alternate # If the cleaned alternate name isn't empty, add it to the list of variants if alternate: name\_variants.append(alternate) # Use re.escape() to safely handle special characters in each name variant for regex name\_variants = [re.escape(name) for name in name\_variants] # Create a regex pattern that matches any of the name variants # \b makes sure we match whole words (e.g., 'Gaza' won't match 'Magazine') regex\_pattern = r"\b(" + "|".join(name\_variants) + r")\b" # Save this pattern and start the match count at 0 in the patterns dictionary patterns[asciiname] = {"pattern": regex\_pattern, "count": 0} is this updated version correct?

**ChatGPT said:**

Awesome — you’re 95% there!  
Let’s review it carefully like a good tutor would, pointing out both the solid bits and tiny improvements you could make:

**✅ What you’ve done well:**

* Corrected reescape to re.escape ✅
* Created name\_variants for both asciiname and alternate names ✅
* Used | to combine them into a regex pattern ✅
* Added word boundaries \b to avoid partial matches ✅
* Built a clean dictionary patterns with regex and count ✅

**🔍 A couple of gentle refinements I’d suggest:**

**1️⃣ Strip whitespace from alternate names**

Right now you’re doing:

python

CopyEdit

alternate = alternate

But you intended to clean it. You should do:

python

CopyEdit

alternate = alternate.strip()

**Why?** Alternate names in the file might have leading/trailing spaces, and those would sneak into your regex otherwise.

**2️⃣ Handle empty alternate names more safely**

Right now:

python

CopyEdit

if alternate:

name\_variants.append(alternate)

That’s fine — but remember that alternate\_names can be an empty string '' (or just commas, like " , "). Stripping will help here too.

**3️⃣ Optional: Case-insensitive matching**

Right now, your regex will only match exact-case versions of the place names.  
If you want to improve recall even more, you might consider adding the re.IGNORECASE flag when using these patterns to search text later.

Example when searching:

python

CopyEdit

re.findall(patterns["Gaza"]["pattern"], text, re.IGNORECASE)

**✅ Clean, fixed version of that small block:**

python

CopyEdit

for alternate in alternate\_list:

alternate = alternate.strip()

if alternate:

name\_variants.append(alternate)

**💡 Final word:**

Yes — your updated version is **basically correct**.  
If you make that one small adjustment above, your recall should improve and your code will be safer against messy data.

Would you like to write a quick test now using a sample text and one of those regex patterns? Great debugging habit before running it on a whole folder 👌