**Mini Project No.2**

**Group Name:** Yasir\_Afreen\_Bushra

**AI Documentation**

**ChatGPT Solution 1:**

patterns = {} rows = data.split("\n") for row in rows[1:]: cells = row.split("\t") ascii\_name = cells[0] patterns[ascii\_name] = 0 SO in our digital humanities class we used a gazetteer to find place name from a collection of articles but there we used only one column named ascii to find placenames but now I want to use name and alternate names columns and I want to built a regex pattern by combining ascii name names and alternate names to increase the recall of the places can you help me with it the above code was used by my teacher

###### **ChatGPT said:**

Absolutely, I can help you with that! You're on the right track — increasing the number of place name variants using the name and alternate names columns will help you catch more matches in your text data.

Here’s how you can modify the code to build a regex pattern that includes ascii, name, and alternate names for each row in your gazetteer:

### **Assumptions**

* Your gazetteer data is tab-separated (.tsv format).
* The columns are: ascii, name, and alternate names.
* The first row is the header.

### **Updated Code**

python

CopyEdit

import re  
  
patterns = {}  
rows = data.split("\n")  
  
header = rows[0].split("\t")  
ascii\_idx = header.index("ascii")  
name\_idx = header.index("name")  
alt\_names\_idx = header.index("alternate names")  
  
for row in rows[1:]:  
 cells = row.split("\t")  
 if len(cells) <= max(ascii\_idx, name\_idx, alt\_names\_idx):  
 continue # Skip incomplete rows  
   
 ascii\_name = cells[ascii\_idx].strip()  
 name = cells[name\_idx].strip()  
 alt\_names = cells[alt\_names\_idx].strip()  
  
 # Split alternate names by comma if they are comma-separated  
 alt\_list = [n.strip() for n in alt\_names.split(",")] if alt\_names else []  
   
 # Combine all names into a set to avoid duplicates  
 name\_variants = set([ascii\_name, name] + alt\_list)  
  
 # Escape each name to safely use in regex (handles special characters)  
 escaped\_names = [re.escape(n) for n in name\_variants if n]  
  
 # Build a regex pattern that matches any of the name variants  
 if escaped\_names:  
 pattern = r"\b(?:{})\b".format("|".join(escaped\_names))  
 patterns[ascii\_name] = re.compile(pattern, re.IGNORECASE)

### **Explanation**

* **set([ascii\_name, name] + alt\_list)** ensures no duplicates.
* **re.escape()** protects special characters like . or ? in place names.
* **\b(?:...)\b** makes sure we only match whole words (e.g., "York" not matching "New Yorkshire").
* **re.IGNORECASE** ensures case-insensitive matching.

Would you like help applying this pattern to match place names in your text corpus too?

4o

##### **You said:**

can you give me a comment on each code explaining what the code does and why it does or why is it necessary so I can clearly understand what are you doingcan you give me a comment on each code explaining what the code does and why it does or why is it necessary so I can clearly understand what are you doing

Cancel

Send

###### **ChatGPT said:**

Absolutely! Here's the updated version of your code, now with detailed inline comments explaining what each line does and **why** it's needed. This version is designed to be both educational and practical for your digital humanities project:

python

CopyEdit

import re # Importing the regular expressions module for pattern matching  
  
patterns = {} # Dictionary to store regex patterns for each place (keyed by ascii name)  
  
rows = data.split("\n") # Split the gazetteer file into rows using newline as the separator  
  
# Extract column indices from the header row to correctly access columns by name  
header = rows[0].split("\t")  
ascii\_idx = header.index("ascii") # Get index of the 'ascii' column  
name\_idx = header.index("name") # Get index of the 'name' column  
alt\_names\_idx = header.index("alternate names") # Get index of the 'alternate names' column  
  
# Loop through all rows except the header  
for row in rows[1:]:  
 cells = row.split("\t") # Split the row into individual cell values by tab  
   
 # Skip the row if it's incomplete (e.g., missing required columns)  
 if len(cells) <= max(ascii\_idx, name\_idx, alt\_names\_idx):  
 continue  
  
 # Extract and clean up the values for each relevant column  
 ascii\_name = cells[ascii\_idx].strip() # Primary name (used as dictionary key)  
 name = cells[name\_idx].strip() # Official or canonical name  
 alt\_names = cells[alt\_names\_idx].strip() # Comma-separated alternate names  
  
 # Split alternate names into a list, trimming spaces; fallback to empty list if blank  
 alt\_list = [n.strip() for n in alt\_names.split(",")] if alt\_names else []  
  
 # Combine all name variants into a set to remove duplicates  
 name\_variants = set([ascii\_name, name] + alt\_list)  
  
 # Escape each name to safely include in regex (e.g., handles special characters like "." or "+")  
 escaped\_names = [re.escape(n) for n in name\_variants if n]  
  
 # Only proceed if there's at least one name to include  
 if escaped\_names:  
 # Create a regex pattern that matches any of the name variants as whole words  
 # \b ensures we only match complete words (not substrings)  
 pattern = r"\b(?:{})\b".format("|".join(escaped\_names))  
  
 # Compile the pattern with case-insensitive matching for flexibility in text  
 patterns[ascii\_name] = re.compile(pattern, re.IGNORECASE)

### **Why Each Part Is Important:**

* **Indexing by header names** ensures the code remains robust even if the column order changes.
* **Stripping** whitespace avoids accidental mismatches.
* **Using a set** removes duplicate names, reducing redundancy in regex.
* **re.escape()** ensures your regex won’t break on names with special characters.
* **Regex pattern with \b(?:...)\b** captures whole-word matches, which is crucial to avoid false positives (e.g., "York" in "NewYork").
* **Case-insensitive matching** increases recall since place names in text can be uppercase, lowercase, or mixed.

Let me know if you want a version of this that also applies the regex to a sample article or dataset.

4o

##### **You said:**

the comment should be before each code and I really donot understand why you have done that can you be more explicit in the comemnt why we are doing that

###### **ChatGPT said:**

Absolutely — here's a version of the code with **explicit, beginner-friendly comments placed above each line**. The comments will explain *what each line does*, *why it’s necessary*, and *how it contributes to your overall goal* (i.e. matching place names in text from a gazetteer). This will be easier to follow, especially for your digital humanities context.

python

CopyEdit

# We need the 're' module to work with regular expressions,  
# which will allow us to build flexible patterns to search for place names in text.  
import re  
  
# Create an empty dictionary to store the compiled regex patterns for each place.  
# The key will be the ascii name (a simple, unique version of the place name),  
# and the value will be a regex pattern that matches any known variant of that place.  
patterns = {}  
  
# We assume 'data' is a string containing the full contents of your gazetteer file,  
# with each row separated by a newline character. This splits it into individual rows.  
rows = data.split("\n")  
  
# The first row usually contains column headers (like "ascii", "name", "alternate names").  
# We split this row by tabs to figure out which column index corresponds to which name.  
header = rows[0].split("\t")  
  
# Find out which index corresponds to each important column so we can use them by name.  
# This makes the code more readable and ensures it still works even if column order changes.  
ascii\_idx = header.index("ascii")  
name\_idx = header.index("name")  
alt\_names\_idx = header.index("alternate names")  
  
# Now we loop through each row after the header (which contains the actual place data).  
for row in rows[1:]:  
 # Split the current row into cells (columns) using the tab character.  
 cells = row.split("\t")  
  
 # Some rows might be incomplete or corrupted. This check avoids index errors  
 # by skipping rows that don't have enough columns.  
 if len(cells) <= max(ascii\_idx, name\_idx, alt\_names\_idx):  
 continue  
  
 # Extract the values from the relevant columns: ascii name, official name, and alternate names.  
 # We use .strip() to remove extra whitespace which might cause matching issues.  
 ascii\_name = cells[ascii\_idx].strip()  
 name = cells[name\_idx].strip()  
 alt\_names = cells[alt\_names\_idx].strip()  
  
 # The alternate names column usually contains a comma-separated list of name variants.  
 # We split these into a list of individual names and remove any extra spaces around them.  
 # If the alternate names field is empty, we just use an empty list.  
 alt\_list = [n.strip() for n in alt\_names.split(",")] if alt\_names else []  
  
 # Combine all possible names for the place: ascii, official name, and all alternates.  
 # We use a set to remove duplicates and avoid repeating the same pattern.  
 name\_variants = set([ascii\_name, name] + alt\_list)  
  
 # Now we escape all name variants using re.escape(), which protects any special characters  
 # like periods, question marks, etc. that might otherwise be misinterpreted as regex syntax.  
 # This ensures we are matching the literal name, exactly as written.  
 escaped\_names = [re.escape(n) for n in name\_variants if n]  
  
 # If there are any valid escaped names, we build a regex pattern.  
 # The pattern uses \b (word boundaries) to ensure we only match whole words.  
 # For example, we want "York" to match "York" but not "NewYork".  
 # The (?:...) part is a non-capturing group that allows matching any of the variants.  
 if escaped\_names:  
 pattern = r"\b(?:{})\b".format("|".join(escaped\_names))  
  
 # Compile the regex pattern for performance and make it case-insensitive.  
 # Case-insensitive means it will match "london", "London", or "LONDON" equally well.  
 patterns[ascii\_name] = re.compile(pattern, re.IGNORECASE)

### **Summary of Why This Code Is Important:**

You're building **a dictionary of regex patterns**, each capable of matching **multiple versions of a place name** in real-world text. This is important in **digital humanities work** because:

* Texts often refer to places in many different ways (e.g., "Bombay", "Mumbai").
* This method helps you **increase recall** — finding more correct place references.
* Using word boundaries (\b) and re.escape() ensures **accuracy** and avoids false positives or broken patterns.

Would you like to see how to apply this dictionary to a sample article or text file to extract place names?

**ChatGPT Solution No.2:**

# Search for each place name using the regex patterns for regex, place in pattern\_to\_place.items(): # Skip if the place has already been counted in this article if place in already\_counted\_places: continue # Use regex to find all occurrences of the place name in the article matches = re.findall(rf"\b(?:{regex})\b", text, flags=re.IGNORECASE) count = len(matches) # Count how many times the place appears # If the place was mentioned, update the total and monthly counts if count > 0: # Update total frequency count for the place frequency\_counter[place] = frequency\_counter.get(place, 0) + count # Update mentions by month for the place if place not in mentions\_per\_month: mentions\_per\_month[place] = {} mentions\_per\_month[place][file\_month] = mentions\_per\_month[place].get(file\_month, 0) + count # Mark this place as counted in the current article already\_counted\_places.add(place) i donot really understand what each code does and why it does can you explain it to me in the form of comments before each code what it does and why it does

###### **ChatGPT said:**

Absolutely! Let me break it down with clear comments explaining both **what** each line of code does and **why** it's being done. Here's the fully explained version:

python

CopyEdit

# Loop through each compiled regex pattern and its corresponding place name  
for regex, place in pattern\_to\_place.items():  
  
 # We only want to count each place \*\*once per article\*\*,   
 # so if we've already counted this place in the current article, we skip it  
 if place in already\_counted\_places:  
 continue  
  
 # Use regex to find all case-insensitive matches of the place name in the article's text  
 # \b ensures we match full words (e.g., "Gaza" not "Magazine")  
 matches = re.findall(rf"\b(?:{regex})\b", text, flags=re.IGNORECASE)  
  
 # Count how many times the place name appears in the article  
 count = len(matches)  
  
 # If the place was mentioned at least once in the article  
 if count > 0:  
  
 # Update the total count of how many times this place has been mentioned across all articles  
 frequency\_counter[place] = frequency\_counter.get(place, 0) + count  
  
 # If this is the first time the place is being added to the monthly dictionary, initialize it  
 if place not in mentions\_per\_month:  
 mentions\_per\_month[place] = {}  
  
 # Update the count of mentions for this place in the current month  
 mentions\_per\_month[place][file\_month] = mentions\_per\_month[place].get(file\_month, 0) + count  
  
 # Mark this place as already counted for this article so it's not double-counted  
 already\_counted\_places.add(place)

### **Summary of the purpose:**

* This code block ensures accurate counting of how many times each place is mentioned **in total** and **per month**, while also avoiding **duplicate counts** within a single article.

Let me know if you'd like a visual or code diagram to go with this explanation!

**ChatGPT Solution No.3:**

Traceback (most recent call last): File "C:/Users/haroon traders/Downloads/FASDH25-portfolio2/scripts/regex\_script\_yasir\_afreen-bushra.py", line 91, in <module> for regex, place in pattern\_to\_place.items(): NameError: name 'pattern\_to\_place' is not defined now i am getting this error

###### **ChatGPT said:**

The new error you're seeing:

pgsql

CopyEdit

NameError: name 'pattern\_to\_place' is not defined

means that your script is trying to use a dictionary named pattern\_to\_place, but that dictionary was never created or defined earlier in the code.

### **🔍 Why this happened:**

In your earlier loop where you build the patterns dictionary:

python

CopyEdit

patterns[ascii\_name] = re.compile(pattern, re.IGNORECASE)

You're compiling regex patterns and storing them with the ascii\_name as the key. But later in your loop (line 91), you're trying to do:

python

CopyEdit

for regex, place in pattern\_to\_place.items():

This implies that you expected a dictionary mapping regex patterns to place names (i.e., the opposite direction of the patterns dictionary), but you never actually created that pattern\_to\_place dictionary.

### **✅ Fix:**

You have **two options** depending on how you want to loop over the data.

#### **✅ Option 1: Update the loop to match your existing dictionary patterns:**

python

CopyEdit

for place, regex in patterns.items():

Since patterns maps ascii\_name → regex, this is the correct way to loop through it.

#### **OR**

#### **✅ Option 2: Create pattern\_to\_place when building patterns:**

If you still want a reverse dictionary for some reason, define it like this:

python

CopyEdit

pattern\_to\_place = {}  
...  
compiled = re.compile(pattern, re.IGNORECASE)  
patterns[ascii\_name] = compiled  
pattern\_to\_place[compiled] = ascii\_name

But this is only necessary if you *really* need regex as the key, which is rare.

### **🚫 Also watch out:**

Later in your script, this line also refers to an undefined variable:

python

CopyEdit

already\_counted\_places.add(place)

But already\_counted\_places hasn't been defined either — you'll need to add:

python

CopyEdit

already\_counted\_places = set()

*at the beginning of the loop for each article*, so it's reset for each file.

Would you like me to help you rewrite the fixed part of your loop completely?

**ChatGPT Solution No.4:**

Traceback (most recent call last): File "C:/Users/haroon traders/Downloads/FASDH25-portfolio2/scripts/regex\_script\_yasir\_afreen-bushra.py", line 95, in <module> if place in already\_counted\_places: NameError: name 'already\_counted\_places' is not defined now I am getting this error

###### **ChatGPT said:**

You're encountering the error:

pgsql

CopyEdit

NameError: name 'already\_counted\_places' is not defined

because your code is trying to use the variable already\_counted\_places, but you haven't created it yet in your script.

### **✅ Fix:**

You need to **initialize** already\_counted\_places **at the beginning of each article's processing** so that it starts fresh for every article (since you only want to count each place once per article).

### **🔧 Where to fix it:**

Find this part of your code:

python

CopyEdit

for filename in os.listdir(articles\_folder):  
 # ...  
 with open(file\_path, encoding="utf-8") as f:  
 text = f.read()

Right **after** reading the text, add this line:

python

CopyEdit

already\_counted\_places = set()

So it should look like this:

python

CopyEdit

with open(file\_path, encoding="utf-8") as f:  
 text = f.read()  
 already\_counted\_places = set()

This will ensure that already\_counted\_places is a new, empty set for each article, avoiding the NameError.

Would you like me to paste a fully fixed version of your article-processing loop?

**ChatGPT Solution No.5:**

Traceback (most recent call last): File "C:/Users/haroon traders/Downloads/FASDH25-portfolio2/scripts/regex\_script\_yasir\_afreen-bushra.py", line 102, in <module> matches = re.findall(rf"\b(?:{regex})\b", text, flags=re.IGNORECASE) File "C:\Users\haroon traders\AppData\Local\Programs\Python\Python313\Lib\re\\_\_init\_\_.py", line 278, in findall return \_compile(pattern, flags).findall(string) File "C:\Users\haroon traders\AppData\Local\Programs\Python\Python313\Lib\re\\_\_init\_\_.py", line 350, in \_compile p = \_compiler.compile(pattern, flags) File "C:\Users\haroon traders\AppData\Local\Programs\Python\Python313\Lib\re\\_compiler.py", line 748, in compile p = \_parser.parse(p, flags) File "C:\Users\haroon traders\AppData\Local\Programs\Python\Python313\Lib\re\\_parser.py", line 980, in parse p = \_parse\_sub(source, state, flags & SRE\_FLAG\_VERBOSE, 0) File "C:\Users\haroon traders\AppData\Local\Programs\Python\Python313\Lib\re\\_parser.py", line 459, in \_parse\_sub itemsappend(\_parse(source, state, verbose, nested + 1, File "C:\Users\haroon traders\AppData\Local\Programs\Python\Python313\Lib\re\\_parser.py", line 865, in \_parse raise source.error("missing ), unterminated subpattern", re.PatternError: missing ), unterminated subpattern at position 2 i am getting this error

###### **ChatGPT said:**

This error:

arduino

CopyEdit

re.PatternError: missing ), unterminated subpattern at position 2

means that **the regex pattern you're passing to re.findall() is broken**, likely due to how you're using it in this line:

python

CopyEdit

matches = re.findall(rf"\b(?:{regex})\b", text, flags=re.IGNORECASE)

But **regex is already a compiled regex object** from this earlier line in your code:

python

CopyEdit

patterns[ascii\_name] = re.compile(pattern, re.IGNORECASE)

### **✅ Fix:**

Since regex is already compiled, **you should NOT wrap it in another string or use re.findall() with flags again**. Instead, just use the regex.findall(text) method directly.

### **Replace this line:**

python

CopyEdit

matches = re.findall(rf"\b(?:{regex})\b", text, flags=re.IGNORECASE)

### **With this:**

python

CopyEdit

matches = regex.findall(text)

### **✅ Bonus Fix: pattern\_to\_place mapping**

If you're also getting an earlier error like:

pgsql

CopyEdit

NameError: name 'pattern\_to\_place' is not defined

You should add this mapping when compiling regexes:

python

CopyEdit

pattern\_to\_place = {}

And update the loop where you compile regexes:

python

CopyEdit

pattern = r"\b(?:{})\b".format("|".join(escaped\_names))  
compiled = re.compile(pattern, re.IGNORECASE)  
patterns[ascii\_name] = compiled  
pattern\_to\_place[compiled] = ascii\_name # This line adds the mapping

Would you like a cleaned-up version of the corrected script?

**ChatGPT Solution No.6:**

##### **You said:**

for pattern, count in patterns.items(): print(f"found {pattern} {count} times") this was a code our teacher gave us printing an output how can i use to print my output in the regex script where I have separately counted the frequences of places separately and their mentions separately

###### **ChatGPT said:**

To help you integrate the for pattern, count in patterns.items(): print(f"found {pattern} {count} times") code into your regex script, I’ll explain how to apply it to your own situation — where you've **counted the frequencies of places separately** and their **mentions separately**.

Let’s assume:

* You have one dictionary for **place frequencies** (e.g., place\_counts)
* And another for **mention frequencies** (e.g., mention\_counts)

Here's how you can print both using similar logic:

python

CopyEdit

# For printing place frequencies  
print("Place Frequencies:")  
for place, count in place\_counts.items():  
 print(f"Found place '{place}' {count} times")  
  
# For printing mention frequencies  
print("\nMention Frequencies:")  
for mention, count in mention\_counts.items():  
 print(f"Found mention '{mention}' {count} times")

This structure allows you to cleanly print both sets of results. If you're using regex to build these dictionaries, make sure they are structured like this:

python

CopyEdit

place\_counts = {  
 "Lahore": 5,  
 "Karachi": 3  
}  
  
mention\_counts = {  
 "city": 4,  
 "village": 2  
}

Would you like help modifying your existing script so it collects and prints these counts properly?

4o

##### **You said:**

# Printing the frequency counts for each place and its monthly mentions to check whether our run whether our script has ran properly for place in frequency\_counter: total = frequency\_counter[place] # Print the total frequency count for the place if it's mentioned at least once if total > 0: print(f"Found {place} {total} times") # Print the frequency count for each month the place was mentioned for month, count in mentions\_per\_month.get(place, {}).items(): if count > 0: print(f" Found {place} {count} times in {month}") I have written it with the help of you earlier do you see an error

###### **ChatGPT said:**

Your script looks **almost perfect** — it's well-structured and logically sound. However, there's **one small typo** in the comment and a minor readability issue in the output formatting.

### **✅ Working Code:**

python

CopyEdit

# Printing the frequency counts for each place and its monthly mentions to check whether our script has run properly  
for place in frequency\_counter:  
 total = frequency\_counter[place]  
  
 # Print the total frequency count for the place if it's mentioned at least once  
 if total > 0:  
 print(f"Found {place} {total} times")  
  
 # Print the frequency count for each month the place was mentioned  
 for month, count in mentions\_per\_month.get(place, {}).items():  
 if count > 0:  
 print(f" Found {place} {count} times in {month}")

### **Minor Fixes:**

1. ✅ Fixed **"our run whether our script has ran"** → **"whether our script has run"**
2. You might consider making the monthly print line more readable, like:

python

CopyEdit

print(f" → In {month}: {count} times")

So it would appear as:

bash

CopyEdit

Found Lahore 5 times  
 → In January: 2 times  
 → In March: 3 times

Would you like help saving this output to a file or visualizing it with a bar chart?

**ChatGPT Solution No.7:**

""" This script is part of our Digital Humanities course Mini Project No. 2, where we explore how to visualize the places mentioned in news articles over time. Using computational tools, we extract toponyms (place names) from each article, map them, and observe how the geographic focus of the news shifts. In this script, we apply regular expressions and a gazetteer to a dataset of 4341 Al Jazeera English articles about the Gaza war, collected by Inacio Vieira. Although the dataset includes older articles, we use a condition to skip those published before the current war (starting November 2023), as our focus is on mapping how the geographic focus of news articles has shifted since the start of the ongoing conflict in Gaza. """ # Importing the regex library to extract place names from text using regular expressions import re # Importing os to interact with the operating system (e.g., accessing files and directories) import os # Importing pandas as pd to store our results in a structured table format (TSV) import pandas as pd # Setting the path to the gazetteer file, which contains place names we want to match gazetteer\_path = "../gazetteers/geonames\_gaza\_selection.tsv" # Setting the path to the folder containing the news articles to be processed articles\_folder = "../articles" # Opening the gazetteer file that contains place names we want to extract from the articles with open(gazetteer\_path, encoding="utf-8") as file: data = file.read() # Reading the entire content of the gazetteer file # Code from below up to line 66 has been fixed with the help of ChatGPT (ChatGPT Solution No.1 in AI Documentation Document) # Creating Dictionary to store regex patterns for each place (keyed by ascii name) patterns = {} # Split the gazetteer file into rows using newline as the separator rows = data.split("\n") # Extract column indices from the header row to correctly access columns by name header = rows[0].split("\t") # Get index of the 'ascii' column ascii\_idx = header.index("asciiname") # Get index of the 'name' column name\_idx = header.index("name") # Get index of the 'alternate names' column alt\_names\_idx = header.index("alternatenames") # Loop through all rows except the header for row in rows[1:]: cells = row.split("\t") # Split the row into individual cell values by tab # Skip the row if it's incomplete (e.g., missing required columns) if len(cells) <= max(ascii\_idx, name\_idx, alt\_names\_idx): continue # Extract and clean up the values for each relevant column ascii\_name = cells[ascii\_idx].strip() # Primary name (used as dictionary key) name = cells[name\_idx].strip() # Official or canonical name alt\_names = cells[alt\_names\_idx].strip() # Comma-separated alternate name # Split alternate names into a list, trimming spaces; fallback to empty list if blank alt\_list = [n.strip() for n in alt\_names.split(",")] if alt\_names else [] # Combine all name variants into a set to remove duplicates name\_variants = set([ascii\_name, name] + alt\_list) # Escape each name to safely include in regex (e.g., handles special characters like "." or "+") escaped\_names = [re.escape(n) for n in name\_variants if n] # Only proceed if there's at least one name to include if escaped\_names: # Create a regex pattern that matches any of the name variants as whole words # \b ensures we only match complete words (not substrings) pattern = r"\b(?:{})\b".format("|".join(escaped\_names)) # Compile the pattern with case-insensitive matching for flexibility in text patterns[ascii\_name] = re.compile(pattern, re.IGNORECASE) # Creating a dictionary named frequency\_counter to track the total mentions of each place name frequency\_counter = {} # Creating a dictionary named mentions\_per\_month to track place name mentions by month mentions\_per\_month = {} # Code from below up to line 118 has been fixed with the help of ChatGPT (see ChatGPT Solution No.2 in AI documentation) # Looping through each article to count place name mentions for filename in os.listdir(articles\_folder): # Extracting the date and month from the filename file\_date = filename[:10] file\_month = file\_date[:7] # Skipping articles published before the Gaza war began (2023-10-07) if file\_date < "2023-10-07": continue # Opening and reading the content of the article file\_path = os.path.join(articles\_folder, filename) with open(file\_path, encoding="utf-8") as f: text = f.read() # Reset counted places for each new article # Code below have been fixed with the help of ChatGPT (see ChatGPT solution No.4 in AI documentation) already\_counted\_places = set() # Loop through each compiled regex pattern and its corresponding place name # Code below have been fixed with the help of ChatGPT (see ChatGPT Solution No.3 in AI documentation) for place, regex in patterns.items(): # We only want to count each place \*\*once per article\*\*, # so if we've already counted this place in the current article, we skip it if place in already\_counted\_places: continue # Use compiled regex to find all case-insensitive matches of the place name in the article's text # Code below have been fixed with the helped with the help of ChatGPT (See ChatGPT Solution No.5 in AI documentation) matches = regex.findall(text) # Count how many times the place name appears in the article count = len(matches) # If the place was mentioned at least once in the article if count > 0: # Update the total count of how many times this place has been mentioned across all articles frequency\_counter[place] = frequency\_counter.get(place, 0) + count # If this is the first time the place is being added to the monthly dictionary, initialize it if place not in mentions\_per\_month: mentions\_per\_month[place] = {} # Update the count of mentions for this place in the current month mentions\_per\_month[place][file\_month] = mentions\_per\_month[place].get(file\_month, 0) + count # Mark this place as already counted for this article so it's not double-counted already\_counted\_places.add(place) # Printing the frequency counts for each place and its monthly mentions to check whether our run whether our script has ran properly # Code below was self written and then was fixed with the help of ChatGPT (See ChatGPT Solution No.6) for place in frequency\_counter: total = frequency\_counter[place] # Print the total frequency count for the place if it's mentioned at least once if total > 0: print(f"Found {place} {total} times") # Print the frequency count for each month the place was mentioned for month, count in mentions\_per\_month.get(place, {}).items(): if count > 0: print(f" Found {place} {count} times in {month}") this is my script and now I want to save its output in the form of a tsv file called regex\_counts.tsv, which contains three columns: placename, month and count. can you me help me do that

###### **ChatGPT said:**

Yes, I can help you with that! You can save the output to a TSV file using pandas. Since you want the file to contain three columns — placename, month, and count — you'll need to convert your mentions\_per\_month dictionary into a format suitable for a DataFrame.

Here’s the code you can **add at the end of your script** (just after your for place in frequency\_counter: block or instead of it, if you're done printing):

python

CopyEdit

# Prepare data for the TSV file  
output\_data = []  
  
# Loop through the mentions\_per\_month dictionary to build rows for the DataFrame  
for place, month\_counts in mentions\_per\_month.items():  
 for month, count in month\_counts.items():  
 output\_data.append({  
 "placename": place,  
 "month": month,  
 "count": count  
 })  
  
# Create a DataFrame from the output data  
df = pd.DataFrame(output\_data)  
  
# Sort the DataFrame by place and month for better readability  
df.sort\_values(by=["placename", "month"], inplace=True)  
  
# Save the DataFrame to a TSV file  
df.to\_csv("regex\_counts.tsv", sep="\t", index=False)  
  
print("Output saved to regex\_counts.tsv")

### **What this does:**

* It builds a list of dictionaries (output\_data), where each dictionary is one row of the TSV.
* It converts the list into a pandas DataFrame.
* It writes the DataFrame to a .tsv file using tab as the separator (sep="\t").

Let me know if you’d also like to include the total counts from frequency\_counter in a separate file.

**ChatGPT Solution No.8:**

import os # create an empty dictionary that will contain our places with their frequencies: places = {} folder = "/content/FASDH25-portfolio2/articles" # loop through all the files in the folder: for filename in os.listdir(folder): # open and read the file: with open(path, encoding="utf-8") as file: text = file.read() # use the nlp pipeline to analyse the text: doc = nlp(text) # select only the entities that are place names: for e in doc.entities: if e.type in ["GPE", "LOC"]: # add 1 to the count of the place in our dictionary # (and/or add the place to the dictionary if it was not there yet): places[e.text] = places.get(e.text, 0) +1 print(places) i want to do this here while I am looping the all the articles I want to use an if condition to incluse only the articles that were published in jan 2024 and want to skip other files and at the end of script I want to print the number of articles published in Jan 2024 and the places dictionary can you help me in writing this code

###### **ChatGPT said:**

Absolutely! Here's how you can modify your script to:

1. **Loop through all files** in a folder.
2. **Check if each article was published in January 2024** (assuming this can be inferred from the filename or content).
3. **Only process those articles**.
4. **Count how many articles were published in January 2024**.
5. **Build the places dictionary** for geographic mentions from those articles.
6. **Print** both the number of qualifying articles and the places with their frequencies.

### **✅ Updated Script with Comments**

python

CopyEdit

import os  
import spacy  
  
# Load spaCy's English language model  
nlp = spacy.load("en\_core\_web\_sm")  
  
# Set the folder path  
folder = "/content/FASDH25-portfolio2/articles"  
  
# Dictionary to hold place name frequencies  
places = {}  
  
# Counter for articles from January 2024  
jan\_2024\_count = 0  
  
# Loop through all files in the folder  
for filename in os.listdir(folder):  
 # Check if the filename or metadata indicates Jan 2024  
 # For example, if the filename includes "2024-01" or "Jan2024"  
 if "2024-01" in filename or "Jan2024" in filename:  
 jan\_2024\_count += 1  
 path = os.path.join(folder, filename)  
  
 # Open and read the file  
 with open(path, encoding="utf-8") as file:  
 text = file.read()  
  
 # Process the text with spaCy  
 doc = nlp(text)  
  
 # Extract geographic entities  
 for ent in doc.ents:  
 if ent.label\_ in ["GPE", "LOC"]:  
 places[ent.text] = places.get(ent.text, 0) + 1  
  
# Print results  
print(f"Number of articles from January 2024: {jan\_2024\_count}")  
print("Places mentioned:")  
print(places)

### **📌 Notes:**

* Make sure that filenames include a date in a consistent format (like 2024-01-15\_article.txt). Adjust the string check ("2024-01" in filename) accordingly.
* If publication dates are **inside the article content**, let me know — you'd need to parse the date from inside the text instead.

Would you like help modifying this for publication dates *within* the file content instead of filenames?

4o

##### **You said:**

i donot want spacy I am doing on collab notebook and I am using stanza already

###### **ChatGPT said:**

Great — since you're using **Stanza** in your Colab notebook, we'll modify your script accordingly. Here's the full working code with the changes:

### **✅ What This Code Does:**

1. Uses **Stanza** (not spaCy) to process text.
2. Only includes articles published in **January 2024** (based on filename).
3. Extracts and counts **place names** (GPE, LOC).
4. Prints:
   1. Number of articles published in Jan 2024.
   2. Dictionary of place frequencies.

### **✅ Final Code Using Stanza:**

python

CopyEdit

import os  
import stanza  
  
# Initialize the Stanza NLP pipeline  
stanza.download("en") # only the first time  
nlp = stanza.Pipeline("en")  
  
# Path to your articles folder  
folder = "/content/FASDH25-portfolio2/articles"  
  
# Dictionary to store places and counter for Jan 2024 articles  
places = {}  
jan\_2024\_count = 0  
  
# Loop through all files in the folder  
for filename in os.listdir(folder):  
 # Check if the file was published in January 2024 (e.g., filename contains '2024-01')  
 if "2024-01" in filename or "Jan2024" in filename:  
 jan\_2024\_count += 1  
 path = os.path.join(folder, filename)  
  
 # Open and read the file  
 with open(path, encoding="utf-8") as file:  
 text = file.read()  
  
 # Process with Stanza  
 doc = nlp(text)  
  
 # Loop through all entities and count GPE and LOC types  
 for sent in doc.sentences:  
 for ent in sent.ents:  
 if ent.type in ["GPE", "LOC"]:  
 places[ent.text] = places.get(ent.text, 0) + 1  
  
# Print results  
print(f"Number of articles from January 2024: {jan\_2024\_count}")  
print("Places mentioned:")  
print(places)

### **📝 Assumption:**

* File names contain '2024-01' or 'Jan2024'. If your date format is different, tell me what it looks like, and I’ll adjust the condition.

Would you like to sort the places dictionary by frequency for better readability?

4o

**ChatGPT Solution No.9:**

so the output this script is giving is quite unclean like When we look at our NER output, it’s quite unclean — for example, the count for Gaza is different from Gaza’s, even though they refer to the same place. Similarly, Red sea and The Red Sea have been counted separately. The same issue happens with United States and US, or Israel and State of Israel. The script we’ve written is meant to clean up these extra named entity names using the concept of normalization, which we learnt in this course. so below I am giving few of methods our teacher has taught us also do cleaning but also go beyond for the issues I am having Other forms of normalization: lower-casing: remove the difference between lower- and upper-cased versions of the same word text = text.lower() whitespace: replace all combinations of whitespace by a single space text = re.sub(r"\s+", " ", text) removing punctuation and/or other non-word characters: text = re.sub(r"[,.?!]", " ", text) text = re.sub(r"\W+", " ", text)

###### **ChatGPT said:**

Thanks for sharing the details — you're absolutely right. Named Entity Recognition (NER) can produce inconsistent outputs for the **same real-world location** due to:

* Capitalization differences
* Variants like “Gaza” vs. “Gaza’s”
* Articles: “The Red Sea” vs. “Red Sea”
* Abbreviations: “US” vs. “United States”
* Formal names: “Israel” vs. “State of Israel”

### **✅ Let’s Fix That with Normalization**

Below is an **improved version of your script**, where we:

1. Use **lowercasing** to treat "Gaza" and "GAZA" as the same.
2. Strip **punctuation and possessives** like 's.
3. Normalize known variations (e.g., “US” → “United States”, “The Red Sea” → “Red Sea”).
4. Collapse **extra whitespace**.

### **✅ Full Cleaned & Normalized Script with Comments**

python

CopyEdit

import os  
import re  
import stanza  
  
# Download and initialize the English NLP pipeline  
stanza.download("en") # only the first time  
nlp = stanza.Pipeline("en")  
  
# Folder containing article files  
folder = "/content/FASDH25-portfolio2/articles"  
  
# Dictionary to store places and a counter for Jan 2024 articles  
places = {}  
jan\_2024\_count = 0  
  
# Define known normalization mappings (expand this as needed)  
normalization\_map = {  
 "us": "united states",  
 "u.s.": "united states",  
 "state of israel": "israel",  
 "the red sea": "red sea",  
 "gaza’s": "gaza",  
 "u.s": "united states",  
 "u.s.a": "united states",  
 "usa": "united states",  
 "u.k.": "united kingdom",  
 "uk": "united kingdom",  
}  
  
# Function to normalize entity text  
def normalize\_entity(text):  
 # Lowercase the text  
 text = text.lower()  
 # Remove possessive 's (e.g., Gaza’s → Gaza)  
 text = re.sub(r"’s|\'s", "", text)  
 # Remove leading "the" for consistency (e.g., "The Red Sea" → "Red Sea")  
 text = re.sub(r"^the ", "", text)  
 # Remove punctuation and non-word characters  
 text = re.sub(r"[,.?!]", "", text)  
 text = re.sub(r"\W+", " ", text)  
 text = re.sub(r"\s+", " ", text).strip() # normalize whitespace  
 # Normalize based on mapping (if present)  
 return normalization\_map.get(text, text)  
  
# Loop through files in the folder  
for filename in os.listdir(folder):  
 # Filter only articles published in January 2024  
 if "2024-01" in filename or "Jan2024" in filename:  
 jan\_2024\_count += 1  
 path = os.path.join(folder, filename)  
  
 # Read the file  
 with open(path, encoding="utf-8") as file:  
 text = file.read()  
  
 # Analyze with Stanza NLP  
 doc = nlp(text)  
  
 # Extract and normalize place entities  
 for sent in doc.sentences:  
 for ent in sent.ents:  
 if ent.type in ["GPE", "LOC"]:  
 normalized = normalize\_entity(ent.text)  
 if normalized: # avoid empty strings  
 places[normalized] = places.get(normalized, 0) + 1  
  
# Print final counts  
print(f"Number of articles from January 2024: {jan\_2024\_count}")  
print("Places mentioned (normalized):")  
print(places)

### **🔍 Example Fixes This Handles:**

|  |  |
| --- | --- |
| **Before** | **After** |
| Gaza’s | gaza |
| US / U.S / USA | united states |
| The Red Sea | red sea |
| State of Israel | israel |
| " Gaza " | gaza |

Would you like me to help sort and print the top 10 most mentioned places as well?

4o

##### **You said:**

now you are merging the script above and this give me a script separately for this

###### **ChatGPT said:**

Got it! Here's a **separate standalone script** that focuses **only on cleaning and normalizing** place names. You can use this after you've already extracted your list of place names and their counts using Stanza.

### **✅ Standalone Script: Normalize Place Names**

python

CopyEdit

import re  
  
# Example: Original dictionary of unnormalized place names  
raw\_places = {  
 "Gaza": 3,  
 "Gaza’s": 2,  
 "The Red Sea": 1,  
 "Red Sea": 1,  
 "US": 4,  
 "U.S.": 2,  
 "USA": 1,  
 "State of Israel": 2,  
 "Israel": 3,  
 "U.K.": 1,  
 "UK": 1  
}  
  
# Define normalization mapping for known variations  
normalization\_map = {  
 "us": "united states",  
 "u.s.": "united states",  
 "u.s": "united states",  
 "usa": "united states",  
 "state of israel": "israel",  
 "the red sea": "red sea",  
 "u.k.": "united kingdom",  
 "uk": "united kingdom",  
 "gaza’s": "gaza"  
}  
  
# Function to normalize entity text  
def normalize\_entity(text):  
 # Lowercase  
 text = text.lower()  
 # Remove possessive ’s or 's  
 text = re.sub(r"’s|\'s", "", text)  
 # Remove "the" from beginning  
 text = re.sub(r"^the ", "", text)  
 # Remove punctuation and extra whitespace  
 text = re.sub(r"[,.?!]", "", text)  
 text = re.sub(r"\W+", " ", text)  
 text = re.sub(r"\s+", " ", text).strip()  
 # Use mapping if available  
 return normalization\_map.get(text, text)  
  
# Create a new cleaned dictionary  
normalized\_places = {}  
  
# Loop through the original dictionary and normalize each key  
for place, count in raw\_places.items():  
 norm = normalize\_entity(place)  
 normalized\_places[norm] = normalized\_places.get(norm, 0) + count  
  
# Print the cleaned, normalized dictionary  
print("Normalized Places:")  
print(normalized\_places)

### **🔍 Output for This Example:**

python

CopyEdit

{  
 'gaza': 5,  
 'red sea': 2,  
 'united states': 7,  
 'israel': 5,  
 'united kingdom': 2  
}

You can plug in your own places dictionary (from the previous Stanza-based script) into this normalization script.

Would you like me to combine this normalization step **into the extraction pipeline** as an all-in-one script, now that you’ve seen it separately?

4o

##### **You said:**

can we do normalization for the case differently you gave me this code somewhere # Step 3: Remove leading 'the' if it appears (to handle "The United States" and "United States") place = re.sub(r"^the\s+", "", place, flags=re.IGNORECASE) # case-insensitive removal of "The"

###### **ChatGPT said:**

Yes — that's a great idea! The line you mentioned:

python

CopyEdit

place = re.sub(r"^the\s+", "", place, flags=re.IGNORECASE)

is **better than text.lower() before removing "The"**, because it:

* Handles both "The Red Sea" and "the Red Sea" cleanly.
* Doesn't force the entire string to lowercase **before** deciding if "The" is at the start — giving us more flexibility.

### **✅ Here's an improved version of the normalization function that separates case normalization logically:**

python

CopyEdit

import re  
  
# Improved normalization function  
def normalize\_entity(place):  
 # Step 1: Remove possessive 's and curly apostrophes like Gaza’s  
 place = re.sub(r"[’']s\b", "", place)  
  
 # Step 2: Remove punctuation (commas, periods, etc.)  
 place = re.sub(r"[,.?!]", "", place)  
 place = re.sub(r"\W+", " ", place)  
  
 # Step 3: Remove leading 'the' if it appears (to handle "The United States" and "United States")  
 place = re.sub(r"^the\s+", "", place, flags=re.IGNORECASE) # case-insensitive removal of "The"  
  
 # Step 4: Normalize extra whitespace  
 place = re.sub(r"\s+", " ", place).strip()  
  
 # Step 5: Lowercase at the very end (only for standardization)  
 place = place.lower()  
  
 # Step 6: Map known variations  
 normalization\_map = {  
 "us": "united states",  
 "u.s.": "united states",  
 "u.s": "united states",  
 "usa": "united states",  
 "state of israel": "israel",  
 "red sea": "red sea",  
 "uk": "united kingdom",  
 "u.k.": "united kingdom",  
 }  
  
 return normalization\_map.get(place, place)

### **🧠 Why this order matters:**

* By **not lowercasing too early**, we can correctly identify "The" at the beginning and remove it cleanly without affecting the rest of the name.
* We **lowercase after cleaning**, so "Red Sea" and "red sea" are still merged.

Would you like me to plug this version into your earlier full pipeline script for final use?

**ChatGPT Solution No.10:**

can you help me write a tsv where I can store my output of normalization and I have to write a tsv file that contains two columns: placename and count

2/2

###### **ChatGPT said:**

Absolutely! Here's a simple Python snippet that writes your normalized\_places dictionary to a .tsv file with two columns: placename and count.

python

CopyEdit

import csv  
  
# Output file path  
output\_file = "normalized\_places.tsv"  
  
# Open the file for writing  
with open(output\_file, mode="w", newline="", encoding="utf-8") as file:  
 writer = csv.writer(file, delimiter="\t")  
  
 # Write the header  
 writer.writerow(["placename", "count"])  
  
 # Write the data rows  
 for place, count in normalized\_places.items():  
 writer.writerow([place, count])  
  
print(f"TSV file saved to {output\_file}")

This will generate a file like:

python-repl

CopyEdit

placename count  
united states 45  
india 30  
united kingdom 12  
...

Would you like to sort the entries alphabetically or by count before saving?

**ChatGPT Solution No.11:**

# Input and output filenames input\_filename = "/content/FASDH25-portfolio2/scripts/ner\_counts.tsv" output\_filename = "/content/FASDH25-portfolio2/scripts/ner\_gazetteer.tsv" # Read place names from ner\_counts.tsv with open(input\_filename, "r", encoding="utf-8") as file: lines = file.readlines() # Skip the header and extract place names because we want to search coordinates for place names place\_names = [line.strip().split("\t")[0] for line in lines[1:]] # Write results to ner\_gazetteer.tsv with open(output\_filename, "w", encoding="utf-8") as out\_file: out\_file.write("Name\tLatitude\tLongitude\n") for name in place\_names: coordinates = get\_coordinates(name) if coordinates: lat = coordinates['latitude'] lon = coordinates['longitude'] out\_file.write(f"{name}\t{lat}\t{lon}\n") else: out\_file.write(f"{name}\tNA\tNA\n") # Display the file with open(output\_filename, encoding="utf-8") as file: print(file.read()) I used this code earlier for geocoding now I did few adjustments for the assignmnet can you make me understand what each line of code is doing and why we are using this code

###### **ChatGPT said:**

Absolutely! Let’s go through your Python code **line by line**, explaining **what it's doing** and **why it’s used**—in the context of your assignment, which appears to involve **geocoding named entities (places)** from a .tsv file.

### **📁 File Paths**

python

CopyEdit

# Input and output filenames  
input\_filename = "/content/FASDH25-portfolio2/scripts/ner\_counts.tsv"  
output\_filename = "/content/FASDH25-portfolio2/scripts/ner\_gazetteer.tsv"

* **What it does:** Stores the paths to your input and output files.
* **Why:** So the program knows **where to read the place names from** and **where to write the geocoded results**.

### **📥 Reading the Input File**

python

CopyEdit

# Read place names from ner\_counts.tsv  
with open(input\_filename, "r", encoding="utf-8") as file:  
 lines = file.readlines()

* **What it does:** Opens the input file and reads all lines into a list called lines.
* **Why:** You need to extract the **place names** from this file for geocoding.

### **🔍 Extracting Place Names**

python

CopyEdit

# Skip the header and extract place names because we want to search coordinates for place names  
place\_names = [line.strip().split("\t")[0] for line in lines[1:]]

* **What it does:**
  + Skips the first line (lines[1:]) because it is assumed to be a **header**.
  + For each remaining line, splits it on the **tab character (\t)** and keeps the **first column** (which is assumed to be the **place name**).
* **Why:** You only care about the **place names** for geocoding—other info (like counts) is irrelevant here.

### **📤 Writing the Output File Header**

python

CopyEdit

# Write results to ner\_gazetteer.tsv  
with open(output\_filename, "w", encoding="utf-8") as out\_file:  
 out\_file.write("Name\tLatitude\tLongitude\n")

* **What it does:** Opens the output file in **write mode** and writes a **header line** for the columns: name, latitude, longitude.
* **Why:** The output should be structured and readable like a table.

### **🌍 Geocoding Each Place Name**

python

CopyEdit

for name in place\_names:  
 coordinates = get\_coordinates(name)

* **What it does:** Loops through each place name and tries to get its coordinates using the function get\_coordinates(name).
* **Why:** You want to find **geographical coordinates** for each place name—this is the core of geocoding.

Note: get\_coordinates(name) is assumed to be a function you defined earlier (or imported) that returns a dictionary with latitude and longitude for a place name, or None if it fails.

### **✅ Writing Coordinates (or NA)**

python

CopyEdit

if coordinates:  
 lat = coordinates['latitude']  
 lon = coordinates['longitude']  
 out\_file.write(f"{name}\t{lat}\t{lon}\n")  
 else:  
 out\_file.write(f"{name}\tNA\tNA\n")

* **What it does:**
  + If coordinates are found: write the name and coordinates.
  + If not found: write the name with NA (Not Available) for both latitude and longitude.
* **Why:** Ensures that every name is recorded in the output, even if geocoding fails.

### **📄 Displaying the Result**

python

CopyEdit

# Display the file  
with open(output\_filename, encoding="utf-8") as file:  
 print(file.read())

* **What it does:** Reopens the output file and prints its contents.
* **Why:** For quick **verification or debugging**, so you can see the final results in the console.

### **Summary: 🧭 What the Code Achieves**

This script takes a list of **named places** (from a .tsv file with NER counts), looks up their **geographical coordinates** (using get\_coordinates()), and writes the results to a new **gazetteer-style file**. This is useful for:

* **Mapping** or **visualizing locations**,
* Adding **geographical context** to named entities,
* Using in **GIS applications** or spatial analysis.

Would you like me to help you write or improve the get\_coordinates() function too?

**ChatGPT Solution No.12:**

import pandas as pd import plotly.express as px # Load data counts = pd.read\_csv("../scripts/regex\_counts.tsv", sep="\t") coords = pd.read\_csv("../gazetteers/geonames\_gaza\_selection.tsv", sep="\t") # Rename for consistency coords = coords.rename(columns={ "asciiname": "placename", "latitude": "latitude", "longitude": "longitude" }) # Merge data = pd.merge(counts, coords, on="placename") # Clean numeric values data["count"] = pd.to\_numeric(data["count"], errors="coerce") data = data.dropna(subset=["count", "latitude", "longitude"]) # Create animated map fig = px.scatter\_map( data, lat="latitude", lon="longitude", hover\_name="placename", size="count", animation\_frame="month", color="count", color\_continuous\_scale=px.colors.sequential.YlOrRd, title="Regex-Extracted Place Names by Month" ) # Save outputs fig.write\_html("regex\_map.html") fig.write\_image("regex\_map.png") # Show the map fig.show() so I took help from you and the slides my teacher gave me to coome up with this code which works perfectly for me can you help me understand what each of the code does so I can write comments before each code explaining what it an why it does

###### **ChatGPT said:**

Here's a detailed breakdown of your code with comments explaining each part:

python

CopyEdit

# Import required libraries  
import pandas as pd # Import pandas for data manipulation and analysis  
import plotly.express as px # Import plotly.express for creating interactive plots  
  
# Load data from CSV files  
counts = pd.read\_csv("../scripts/regex\_counts.tsv", sep="\t") # Load the regex counts data  
coords = pd.read\_csv("../gazetteers/geonames\_gaza\_selection.tsv", sep="\t") # Load the geographical coordinates data  
  
# Rename columns for consistency to match the columns in the counts dataset  
coords = coords.rename(columns={ # Rename columns for better readability  
 "asciiname": "placename", # Rename 'asciiname' to 'placename' to match the 'counts' dataset  
 "latitude": "latitude", # Keep 'latitude' as is  
 "longitude": "longitude" # Keep 'longitude' as is  
})  
  
# Merge the counts data with the geographical coordinates data based on the 'placename' column  
data = pd.merge(counts, coords, on="placename") # Merge the two datasets based on the 'placename' column  
  
# Convert the 'count' column to numeric values, coercing errors to NaN (if any non-numeric values are present)  
data["count"] = pd.to\_numeric(data["count"], errors="coerce") # Convert the 'count' column to numeric format  
data = data.dropna(subset=["count", "latitude", "longitude"]) # Drop rows with missing 'count', 'latitude', or 'longitude' values  
  
# Create an animated map using Plotly Express  
fig = px.scatter\_map( # Create a scatter map with Plotly Express  
 data, # The data to plot  
 lat="latitude", # Use 'latitude' for the y-axis (latitude)  
 lon="longitude", # Use 'longitude' for the x-axis (longitude)  
 hover\_name="placename", # Display the 'placename' when hovering over a point  
 size="count", # Use the 'count' column to determine the size of the points on the map  
 animation\_frame="month", # Create an animation based on the 'month' column  
 color="count", # Color the points based on the 'count' value  
 color\_continuous\_scale=px.colors.sequential.YlOrRd, # Set the color scale to 'YlOrRd' (yellow-orange-red)  
 title="Regex-Extracted Place Names by Month" # Set the title of the map  
)  
  
# Save the animated map as HTML and PNG files  
fig.write\_html("regex\_map.html") # Save the interactive map as an HTML file  
fig.write\_image("regex\_map.png") # Save the map as a PNG image  
  
# Display the map  
fig.show() # Show the interactive map in the notebook

### **What each part does:**

1. **Import Libraries**:
   1. pandas is used for handling data in tabular format.
   2. plotly.express is used for creating interactive visualizations.
2. **Load Data**:
   1. pd.read\_csv() loads your data from .tsv files into DataFrames. These files contain counts of regex matches and geographical information like placenames and coordinates.
3. **Renaming Columns**:
   1. You rename columns in the coords DataFrame for consistency with the counts DataFrame. This is important for the merging step.
4. **Merge Data**:
   1. The pd.merge() function combines the two DataFrames (counts and coords) by matching the placenames in both DataFrames, allowing you to associate each placename with its count and geographic location.
5. **Clean Data**:
   1. The pd.to\_numeric() function ensures that the 'count' column contains numeric values, and the errors="coerce" argument converts any non-numeric values into NaN.
   2. dropna() removes rows with missing values in important columns (count, latitude, or longitude) to ensure the data is clean for plotting.
6. **Create Animated Map**:
   1. px.scatter\_map() creates a scatter plot on a map, with each point representing a placename. The points are sized according to the 'count' column and are colored based on the same 'count' column.
   2. animation\_frame="month" allows the map to animate over time, showing how the data evolves by month.
   3. The color scale YlOrRd provides a yellow-to-red gradient, where higher counts are shown in red.
7. **Save Outputs**:
   1. fig.write\_html() and fig.write\_image() save the interactive map as an HTML file and a static PNG image, respectively.
8. **Show the Map**:
   1. fig.show() displays the map in your environment, allowing you to interact with it.

Each section is designed to clean, merge, visualize, and save your data in an interactive format. Adding comments will help you understand the flow and purpose of each line.

**ChatGPT Solution No.13:**

import pandas as pd import plotly.express as px # Load data counts = pd.read\_csv("../scripts/ner\_counts.tsv", sep="\t") coords = pd.read\_csv("../scripts/NER\_gazetteer.tsv", sep="\t") # Rename columns in coords to match counts and plotting requirements coords = coords.rename(columns={ "Name": "placename", "Latitude": "latitude", "Longitude": "longitude" }) # Merge data on 'placename' data = pd.merge(counts, coords, on="placename") # Ensure 'count' is numeric and drop rows with missing values data["count"] = pd.to\_numeric(data["count"], errors="coerce") data = data.dropna(subset=["count", "latitude", "longitude"]) # Plot map fig = px.scatter\_map( data, lat="latitude", lon="longitude", hover\_name="placename", size="count", color="count", title="NER-extracted Place Names (Jan 2024)", ) # Save outputs fig.write\_html("NER\_map.html") fig.write\_image("NER\_map.png") #show the map fig.show() so with the help of ChatGPT and our teacher slide we craeted this code for our NER counts so what we want to do is to you to explain us this code very simply so that we can generate comments explaining what the code does and why we use that code

###### **ChatGPT said:**

Absolutely! Let's break down the code step by step and explain what each part does in simple terms.

### **1. Importing Libraries**

python

CopyEdit

import pandas as pd  
import plotly.express as px

* **pandas**: A powerful Python library used to work with data in the form of tables (called DataFrames). It allows for easy data manipulation.
* **plotly.express**: A library that makes it simple to create interactive plots, like maps, charts, and graphs.

### **2. Loading Data**

python

CopyEdit

counts = pd.read\_csv("../scripts/ner\_counts.tsv", sep="\t")  
coords = pd.read\_csv("../scripts/NER\_gazetteer.tsv", sep="\t")

* **pd.read\_csv()**: This function reads data from a file (CSV or TSV format). In this case:
  + **ner\_counts.tsv** contains the count of named entities recognized (NER counts).
  + **NER\_gazetteer.tsv** contains geographic information (place names, latitudes, longitudes).
* **sep="\t"**: This tells pandas that the files are tab-separated (TSV format).

### **3. Renaming Columns in coords to Match counts**

python

CopyEdit

coords = coords.rename(columns={  
 "Name": "placename",  
 "Latitude": "latitude",  
 "Longitude": "longitude"  
})

* Here, we rename the columns in the coords DataFrame to standardize the column names:
  + **"Name"** becomes **"placename"**.
  + **"Latitude"** becomes **"latitude"**.
  + **"Longitude"** becomes **"longitude"**.
* This is done so that when we merge the data, the columns have the same name, making it easier to join them.

### **4. Merging Data on placename**

python

CopyEdit

data = pd.merge(counts, coords, on="placename")

* **pd.merge()**: This combines two DataFrames (counts and coords) into one.
* We are merging them on the **placename** column, meaning that the rows with the same place name from both datasets will be combined together.

### **5. Cleaning Data**

python

CopyEdit

data["count"] = pd.to\_numeric(data["count"], errors="coerce")  
data = data.dropna(subset=["count", "latitude", "longitude"])

* **pd.to\_numeric()**: This converts the **count** column to numbers. If there are any non-numeric values, they will be turned into **NaN** (Not a Number) using the argument errors="coerce".
* **dropna()**: This removes any rows where there are missing values in the **count**, **latitude**, or **longitude** columns, ensuring that the data used for plotting is complete.

### **6. Plotting the Map**

python

CopyEdit

fig = px.scatter\_map(  
 data,  
 lat="latitude",  
 lon="longitude",  
 hover\_name="placename",  
 size="count",  
 color="count",  
 title="NER-extracted Place Names (Jan 2024)",  
)

* **px.scatter\_map()**: This function creates an interactive map.
  + **data**: The data being plotted.
  + **lat** and **lon**: These specify which columns contain the latitude and longitude for each place.
  + **hover\_name="placename"**: When you hover over a point on the map, it will show the **placename**.
  + **size="count"**: The size of each point on the map will be determined by the **count** value (the number of times that place was mentioned).
  + **color="count"**: The color of each point will also be based on the **count**, creating a color gradient.
  + **title="NER-extracted Place Names (Jan 2024)"**: This sets the title of the map.

### **7. Saving the Map**

python

CopyEdit

fig.write\_html("NER\_map.html")  
fig.write\_image("NER\_map.png")

* **write\_html()**: This saves the map as an interactive HTML file.
* **write\_image()**: This saves the map as a static PNG image.

### **8. Displaying the Map**

python

CopyEdit

fig.show()

* **fig.show()**: This displays the map in the output (in a Jupyter notebook or browser).

### **Summary**

This code:

* Loads data on place names, their counts, and geographic locations.
* Cleans and merges the data.
* Creates an interactive map where each place is represented by a point. The size and color of the points depend on how often the place is mentioned.
* Saves the map as both an HTML (interactive) and a PNG (static) file.
* Displays the map to the user.

### **Comments**

Here are simple comments you can add to the code:

python

CopyEdit

# Importing necessary libraries for data manipulation and plotting  
import pandas as pd  
import plotly.express as px  
  
# Loading the NER counts and geographic data from TSV files  
counts = pd.read\_csv("../scripts/ner\_counts.tsv", sep="\t")  
coords = pd.read\_csv("../scripts/NER\_gazetteer.tsv", sep="\t")  
  
# Renaming columns in coords to match the ones in counts for easier merging  
coords = coords.rename(columns={  
 "Name": "placename",  
 "Latitude": "latitude",  
 "Longitude": "longitude"  
})  
  
# Merging the two dataframes on the common 'placename' column  
data = pd.merge(counts, coords, on="placename")  
  
# Converting 'count' to numeric and removing rows with missing data  
data["count"] = pd.to\_numeric(data["count"], errors="coerce")  
data = data.dropna(subset=["count", "latitude", "longitude"])  
  
# Creating an interactive map using plotly, with place names, counts, and geographic coordinates  
fig = px.scatter\_map(  
 data,  
 lat="latitude", # Latitude column  
 lon="longitude", # Longitude column  
 hover\_name="placename", # Display the place name on hover  
 size="count", # Size of points based on the count  
 color="count", # Color of points based on the count  
 title="NER-extracted Place Names (Jan 2024)" # Title of the map  
)  
  
# Saving the map as an interactive HTML file and static PNG image  
fig.write\_html("NER\_map.html")  
fig.write\_image("NER\_map.png")  
  
# Displaying the map  
fig.show()

This should help you understand and explain the code clearly!