ChatGPT

Conversation 1

i am a digital humanities student and i have to write a code to extract place names from articles with the help of NER method. so far i have extracted the place names after jan 2024 and counted the number of times the place has been mentioned now i have to clean up the named entity names to check if the data contains any duplicate of the names and merge it using conditions can you explain and give me examples how can i do that

**ChatGPT said:**

Great, you’ve made solid progress!

Here’s what you need to do next, broken into **clear steps** with an **example** using Python and spaCy (assuming you are using spaCy or a similar NER tool):

**Step 1: Extract place names (you’ve done this)**

Example result (after NER + filtering after Jan 2024):

python

CopyEdit

places = ['New York', 'NYC', 'New York City', 'Los Angeles', 'LA', 'L.A.', 'San Francisco', 'SF', 'San Fran']

counts = {'New York': 5, 'NYC': 3, 'New York City': 2, 'Los Angeles': 4, 'LA': 2, 'L.A.': 1, 'San Francisco': 3, 'SF': 2, 'San Fran': 1}

**Step 2: Normalize names**

Many places have variations — e.g., **NYC**, **New York City**, and **New York** all refer to the same place.  
You can **create a mapping dictionary** to unify these.

python

CopyEdit

name\_mapping = {

'NYC': 'New York',

'New York City': 'New York',

'LA': 'Los Angeles',

'L.A.': 'Los Angeles',

'SF': 'San Francisco',

'San Fran': 'San Francisco'

}

**Step 3: Merge counts**

Loop over your counts and **merge duplicates** using the mapping.

python

CopyEdit

merged\_counts = {}

for name, count in counts.items():

# Map to unified name if exists, else keep original

unified\_name = name\_mapping.get(name, name)

# Add to merged count

if unified\_name in merged\_counts:

merged\_counts[unified\_name] += count

else:

merged\_counts[unified\_name] = count

**Step 4: Check result**

python

CopyEdit

print(merged\_counts)

Expected output:

bash

CopyEdit

{

'New York': 10,

'Los Angeles': 7,

'San Francisco': 6

}

**Optional Cleanup**

If you want to also **standardize case** (e.g., ‘new york’ vs. ‘New York’), you can:

python

CopyEdit

standardized\_name = unified\_name.lower().title()

**Summary of logic**

✅ Build a manual or semi-automated **mapping** of duplicates  
✅ Use .get() to replace names with unified labels  
✅ Aggregate counts into a **new dictionary**

If you want, I can help you write a **ready-to-run script** using your actual data format — do you want to share a small sample of your extracted place names and counts?

4o

**You said:**

is there any other way to do it without manually typing them

**ChatGPT said:**

Yes — **there are semi-automated or automated ways** to reduce manual mapping, though they won’t be 100% perfect and often need some fine-tuning.

Here are **three approaches** you can use **without manually typing mappings**:

**1️⃣ String similarity matching (e.g., fuzzy matching)**

You can use the fuzzywuzzy or rapidfuzz library to check if two names are “close enough” (say, ≥90% similar) and treat them as duplicates.

Example:

python

CopyEdit

from rapidfuzz import process

place\_list = list(counts.keys())

merged\_counts = {}

for name, count in counts.items():

match, score, \_ = process.extractOne(name, merged\_counts.keys(), score\_cutoff=90)

if match:

merged\_counts[match] += count

else:

merged\_counts[name] = count

✅ **Pros:** No need to define mappings manually.  
⚠ **Cons:** Might incorrectly merge names that are similar but actually different (e.g., “Newark” vs. “New York”).

**2️⃣ Use a geographic database (like GeoNames or spaCy’s GPE)**

Instead of relying only on names, you can **resolve entities** against a standard geographic database or API. For example:

* Use a **GeoNames API** to resolve different variants to a unique ID.
* Use spaCy’s entity linking (if available in your model) to connect “NYC” and “New York” to the same real-world place.

✅ **Pros:** More robust; based on real geographic data.  
⚠ **Cons:** Requires internet/API access and setup.

**3️⃣ Normalize using lowercasing, removing punctuation, and token rules**

Example:

python

CopyEdit

import re

def normalize\_name(name):

name = name.lower()

name = re.sub(r'[^\w\s]', '', name) # remove punctuation

name = name.replace('city', '').strip()

return name

merged\_counts = {}

for name, count in counts.items():

norm\_name = normalize\_name(name)

if norm\_name in merged\_counts:

merged\_counts[norm\_name] += count

else:

merged\_counts[norm\_name] = count

✅ **Pros:** Simple and automatic; no libraries needed.  
⚠ **Cons:** May over-simplify and merge unintended names.

**Which one should you use?**

* **If your dataset is small or medium-sized:** A fuzzy match or simple normalization should work well.
* **If you need high accuracy or have many international place names:** Consider integrating a geographic resolver or database.

If you tell me **what libraries you are allowed to use** (or prefer) and maybe share a **small sample of your place list**, I can help write the most suitable code for your project. Would you like me to draft a sample script?

4o

**You said:**

the code that you provided is quite difficult but i undersatnd that i have to do it in this way def normalize\_name(name): name = name.lower() name = re.sub(r'[^\w\s]', '', name) # remove punctuation name = name.replace('city', '').strip() return name

**ChatGPT said:**

Good job picking that out — yes, that’s the **simplest and most beginner-friendly approach** you can start with!

Let me **walk you through** how it works, step by step:

**🔍 Explanation of the normalize\_name function**

python

CopyEdit

def normalize\_name(name):

name = name.lower() # make everything lowercase → 'New York' → 'new york'

name = re.sub(r'[^\w\s]', '', name) # remove punctuation → 'l.a.' → 'la'

name = name.replace('city', '').strip() # remove common words → 'new york city' → 'new york'

return name

So:

* "New York" → "new york"
* "New York City" → "new york"
* "L.A." → "la"
* "San Fran" → "san fran" (you might need to add more replacements if needed)

**🔄 How to apply it to your place counts**

Here’s a **full example script** using your counts dictionary:

python

CopyEdit

import re

# Original place counts

counts = {

'New York': 5,

'NYC': 3,

'New York City': 2,

'Los Angeles': 4,

'LA': 2,

'L.A.': 1,

'San Francisco': 3,

'SF': 2,

'San Fran': 1

}

# Normalize function

def normalize\_name(name):

name = name.lower()

name = re.sub(r'[^\w\s]', '', name) # remove punctuation

name = name.replace('city', '').strip()

return name

# Merge counts

merged\_counts = {}

for name, count in counts.items():

norm\_name = normalize\_name(name)

if norm\_name in merged\_counts:

merged\_counts[norm\_name] += count

else:

merged\_counts[norm\_name] = count

print(merged\_counts)

**🏁 Expected output**

bash

CopyEdit

{

'new york': 10,

'nyc': 3,

'los angeles': 7,

'la': 2,

'san francisco': 6,

'san fran': 1

}

**⚠ Note**

Notice that:

* NYC and New York are still counted separately
* LA and Los Angeles are still counted separately

If you want **further merging** (e.g., treating 'nyc' and 'new york' as the same), you **will** need:  
✅ A mapping dictionary **OR**  
✅ Fuzzy matching (more advanced)

Would you like me to write a version that **also merges 'nyc' → 'new york' and 'la' → 'los angeles' automatically** without you typing a manual list?

4o

**You said:**

i ran the code and i have been waiting for it to run for more than 15 mins. is it talking time because of the code or is there any other issue

**ChatGPT said:**

No — the simple normalization + merging code I gave **should run almost instantly**, even on thousands of entries.

If it’s been running for **15+ minutes**, something else is likely causing the delay.

Here are **possible causes** you should check:

✅ **1️⃣ Infinite loop or very large dataset**  
Did you accidentally nest the loop or call the script recursively?  
Double-check your loop:

python

CopyEdit

for name, count in counts.items():

# Only loops over the dictionary once

Also, how big is your counts dictionary? If it has **millions** of entries, it will naturally take longer, though still probably not 15+ minutes.

✅ **2️⃣ Heavy preprocessing or external calls**  
If you accidentally added something like:

* A time.sleep() or input() inside the loop
* An API or file call inside the loop

…it will dramatically slow things down.

✅ **3️⃣ Memory or environment issues**  
If you’re running this on:

* A **low-RAM machine** (especially with big data)
* A **restricted online notebook** (like Google Colab with memory limits)

…it might freeze or crash.

✅ **4️⃣ Wrong dataset fed in**  
Did you pass in the full article texts or NER objects instead of just the **counts dictionary**?  
You should only loop over:

python

CopyEdit

counts = {'New York': 5, 'NYC': 3, ...}

not over raw entity objects or full texts.

**🔧 How to check**

Try adding this **debug print** before the loop:

python

CopyEdit

print("Number of entries:", len(counts))

If you see something like **10 million entries**, we’ll need to optimize.

If the number is reasonable (e.g., hundreds or thousands), the script should finish in under a second.

Would you like to paste here:  
✅ The **size** of your dataset  
✅ A **small sample** of the input you’re running it on

so I can help you debug it directly?

4o

**You said:**

normalization is not the only code i have other codes before and after that

**ChatGPT said:**

Ah — thanks, that’s important context!

If your **overall script** (including the NER extraction, filtering, normalization, and merging) is running **slowly**, the bottleneck is likely **before or after** the normalization step, not in the simple dictionary loop I provided.

Here’s what you can do to narrow it down:

**🔍 Step 1: Add timing checkpoints**

You can check **which part is slow** by adding timing markers.

Example:

python

CopyEdit

import time

start = time.time()

# --- NER extraction ---

# your NER code here

print("NER extraction done in", time.time() - start, "seconds")

start = time.time()

# --- Filtering after Jan 2024 ---

# your filtering code here

print("Filtering done in", time.time() - start, "seconds")

start = time.time()

# --- Normalization and merging ---

# normalization + merge loop here

print("Normalization and merging done in", time.time() - start, "seconds")

This will tell you **which block is causing the delay**.

**🔍 Step 2: Check the NER part**

The most common slowdowns happen in:  
✅ Processing **very large texts** (many long articles)  
✅ Using a **heavy model** (large spaCy models like en\_core\_web\_trf instead of en\_core\_web\_sm)  
✅ Running NER **inside a loop** on each article, rather than batching

If you’re running something like:

python

CopyEdit

for article in articles:

doc = nlp(article)

…it’s **much slower** than:

python

CopyEdit

docs = list(nlp.pipe(articles))

**🔧 What can you do?**

* Switch to nlp.pipe() to process many articles **faster**
* Reduce model size if you don’t need high accuracy (en\_core\_web\_sm is much faster than transformer models)
* If the bottleneck is **after** NER (e.g., in file writing), look there

If you paste a **brief overview of your script sections or main blocks**, I can help you figure out where the slowdown is — want to do that?