1. **Help for defining start and end date for n-gram date**

ChatGPT

**You said:**

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:**

Great, Ulya! I'd be happy to guide you like a tutor—helping you understand the *why* and *how*, not just giving the *what*. To get started, tell me where you're currently stuck or what specific part of your Digital Humanities Python project you'd like help with. Is it about text processing, visualization, regular expressions, data cleaning, or something else?

Some general advice to begin with:

**📌 1. Break the Problem into Steps**

Instead of looking at the whole task at once, divide it:

* What data do you have?
* What transformation or analysis do you need?
* What should the output look like?

Write down each step in comments before coding it.

**🔁 2. Use Small Print Checks**

When working with new code (e.g., regex, tokenizing, or filtering), use print() or df.head() after each step to verify that the output is what you expected. Debugging gets 10× easier this way.

**🧹 3. Clean Before You Analyze**

If your text or data isn't clean (extra whitespace, punctuation, etc.), your analysis (like n-gram counts or entity recognition) will be inaccurate. Do basic cleanup first using .lower(), .strip(), or regex to remove non-text elements.

**📊 4. Choose the Right Visualization**

When plotting:

* **Bar charts** are great for frequency counts.
* **Line plots** show changes over time.
* **Word clouds** are tempting, but they rarely show true proportions.

Ask yourself: *What story should this plot tell?*

**🧠 5. Think Like a Researcher**

DH isn't just about coding—it's about insight. Ask:

* What does this frequency spike mean in context?
* Is there a historical or media pattern behind the shift?

Your code supports your interpretation—not the other way around.

Tell me where you’re at in your project right now, and I’ll help you tackle the next hurdle thoughtfully.

**You said:**

i am working on n-gram 1, and i am planning to filtered out two categories of words like words related to conflict and words related to peace, I have already made two sets of lists of words, and I know if I assign a new list, for example all words and add those previous two sets it will give me a complete set on which I can work, I know that this code "df = df[df['1-gram'].isin(all\_words)]" will help me to filter out the datagram, it gave me the output but I want to restrict these words from 0ct 7 2023 to 0ct 7 2024

**2/2**

**ChatGPT said:**

That's a great plan, Ulya! You're thinking exactly like a DH researcher—categorizing words by theme and filtering them temporally. You're almost there. Let's break it down into steps and refine your approach without handing you the full code.

**🧩 Step-by-Step Tutor Advice**

**✅ You Already Have:**

* Two lists of words: one for **conflict**, one for **peace**.
* You've combined them into a single all\_words list.
* You’ve used df[df['1-gram'].isin(all\_words)] to filter rows that match these words. Perfect!

**⏳ Now, You Want:**

To **restrict your data** to the time period between **October 7, 2023, and October 7, 2024**.

**🧠 Think Through These Key Questions:**

**1. Is your date column in datetime format?**

If not, you need to convert it using pd.to\_datetime() first.

✅ Tip: Always convert date strings early so that filtering is accurate.

**2. How do you filter a DataFrame by date?**

You’ll need to use logical conditions like:

python

Copy code

df[(df['date'] >= start\_date) & (df['date'] <= end\_date)]

💡 Remember: start\_date and end\_date should be datetime objects.

**📌 Suggested Pseudocode Approach (Not Final Code!)**

python

Copy code

*# 1. Convert the date column if needed* df['date'] = pd.to\_datetime(df['date']) *# 2. Define your start and end dates* start\_date = pd.Timestamp('2023-10-07') end\_date = pd.Timestamp('2024-10-07') *# 3. Apply both filters: word and date* filtered\_df = df[ (df['1-gram'].isin(all\_words)) & (df['date'] >= start\_date) & (df['date'] <= end\_date) ]

**🧪 Final Check:**

After this, use filtered\_df.head() to ensure your output makes sense:

* Are the words from your list?
* Are the dates in the correct range?

Let me know if your date column has a different name or format, or if you're planning to compare **peace vs. conflict frequency** separately later. I can help you set up that logic next.

Top of Form

Tools

Bottom of Form

ChatGPT can make mistakes. Check important info.

1. **Categorizing the code**

i ran a set of code but it is giving me graph between total counts and words, but not comparing the two word list, I think I need to categosise the word list so it gave me a comparison between them, how to categorise these two sets of word list

**ChatGPT said:**

Excellent intuition—you’re right! If your current graph is just showing total counts by word, it’s treating all your words as one group. To **compare peace vs. conflict words**, you need to **label** or **categorize** each word with its group before plotting.

Let me guide you toward that:

**🧠 Ask Yourself:**

**How can I add a column that tells me whether a word is related to "peace" or "conflict"?**

Your goal is to annotate each row with a *category label*, based on whether the '1-gram' word is in the peace\_words list or the conflict\_words list.

**🧩 Think in Steps:**

1. You already have peace\_words and conflict\_words as Python lists or sets.
2. You want to **map** each row’s '1-gram' value to a category:
   * If it’s in peace\_words, label it 'peace'
   * If in conflict\_words, label it 'conflict'
   * (Optional: if not in either, label it 'other' or leave as NaN)

🔎 **What Python function or method allows you to apply a custom label to each row based on a condition?**

(Hint: think about row-wise operations... maybe using .apply() or even a lambda function.)

**🛠 Small Exercise:**

Try writing a function that takes a word and returns the label 'peace' or 'conflict'. You can test it on a few words before applying it to the whole DataFrame.

Then think:

How do I use this function to create a new column in df?

Once you’ve labeled the words, you can easily group by the 'category' column and compare total counts for each group.

Let me know how far you get with this, and I’ll nudge you again if you need it!

**You said:**

oky i will write the script here tell me if this is write or wrong df ['catrgory'] = df['1-gram'].apply( if conflict\_realted\_terms level it 'conflict' , if in peace\_realated\_term label it 'conflict', else as 'other')

**ChatGPT said:**

You're on the right path conceptually, Ulya, but your **syntax** needs a bit of tweaking to be valid Python. You're trying to write logic inside .apply(), but it needs to be wrapped in a function (or lambda), and your second condition has a small bug—you're labeling both as 'conflict'.

Let's walk through how to **think it through and correct it**, not just give you the final line.

**🔍 Ask Yourself:**

1. **How do I check if a word is in a list or set?**  
   → if word in peace\_related\_terms
2. **How do I apply that check to each row in a column?**  
   → Use .apply(lambda x: ...)
3. **How do I assign the result to a new column?**  
   → df['category'] = ...

**🛠 Your next move:**

Try wrapping that logic in a lambda function like this pseudocode structure:

python

Copy code

df['category'] = df['1-gram'].apply(

lambda word: 'conflict' if word in conflict\_related\_terms

else 'peace' if word in peace\_related\_terms

else 'other'

)

🔁 Think before running:

* Are your lists peace\_related\_terms and conflict\_related\_terms sets? (If not, consider making them sets for faster lookup.)
* Are the words in your dataframe all lowercase like your lists? (Use .lower() if needed for consistency.)

Give it a try! Let me know what happens—do the new categories look right when you check df['category'].value\_counts()?

**You said:**

what is this lambda doing

**ChatGPT said:**

Great question—understanding lambda will unlock so much power in Python for you!

**🔎 So, What is a lambda?**

A lambda is a way to write **a small, unnamed function** in one line. It's useful when you want to apply a function to something **on the fly**, like inside .apply().

**🧠 Here's the Structure:**

python

Copy code

lambda input: output

You're basically saying:

"Take input, do something to it, and return the output."

**💡 In Your Case:**

python

Copy code

lambda word: 'conflict' if word in conflict\_related\_terms

else 'peace' if word in peace\_related\_terms

else 'other'

Break this down:

* **word** is each value from the '1-gram' column.
* The function checks:
  + If word is in your conflict list → label it 'conflict'.
  + Else if it’s in the peace list → label it 'peace'.
  + Else → label it 'other'.

**🔁 What Happens Inside .apply()?**

When you write:

python

Copy code

df['category'] = df['1-gram'].apply(lambda word: ...)

It’s like saying:

“Go through every value in the '1-gram' column, one by one, run this lambda function on it, and store the result in the new 'category' column.”

Try printing a few examples to see how it works:

python

Copy code

print((lambda word: 'conflict' if word in conflict\_related\_terms else 'peace' if word in peace\_related\_terms else 'other')('ceasefire'))

Would it label 'ceasefire' correctly?

Let me know if you want to refactor it into a named function too—sometimes that’s better for longer or more readable logic.

Heading 3 frequency comparision

for groupby frequencies we learn this code in class grouped\_df = (df.groupby('1-gram', as\_index=False)["count-sum"].sum() .sort\_values('count-sum', ascending=False)) i tried to update this according to my datagram, tell me if I am doing it correct monthly\_count = df.groupby('date') ['count-sum'].sum()

**ChatGPT said:**

You're almost there, Ulya—you've got the idea **exactly right**, just need to think about what your **goal** is and whether the **output** fits that goal.

Let’s break it down together:

**✅ What Your Code Does:**

python

Copy code

monthly\_count = df.groupby('date')['count-sum'].sum()

This groups your data by **individual dates** (likely daily), not *months*. So, if you're aiming for **monthly frequencies**, this won’t quite give you that.

**❓Ask Yourself:**

Do I want to group by **day**, or by **month** (e.g., Oct 2023, Nov 2023, etc.)?

**💡 If You Want Monthly Grouping:**

Think about whether your date column is in datetime format. If it is, then…

🧠 What attribute can you use from a datetime object to extract year and month?

(Hint: .dt.to\_period('M') or extract year/month separately)

**🧪 Try This Thought Exercise:**

Can you create a new column called 'month' using:

python

Copy code

df['month'] = df['date'].dt.to\_period('M')

Then try grouping by that new 'month' column.

**Bonus Thinking:**

After getting monthly\_count, consider:

* Do you want to compare **peace vs conflict** by month?
* If yes, how could you include 'category' in the groupby()?

Try playing with:

python

Copy code

df.groupby(['month', 'category'])['count-sum'].sum()

Let me know what you try next, and I’ll keep guiding. You're really close!

**Heading 4 Finding frequencies by groupby**

i know putting relative frequency on df but I need to compute it on groupby I am currently using this script df = df.merge(month\_total, on='date') df['relative\_freq'] = df['count-sum'] / df['total\_per\_month'] give me idea how to do it on group by

**ChatGPT said:**

Great question! When you want to calculate relative frequency **within groups**, groupby is definitely the way to go.

Here’s a conceptual hint without full code:

1. Use groupby on your column that defines the groups (like 'date' or 'month').
2. For each group, compute the total count (or sum) of the relevant column.
3. Then **divide each row’s count by the total count of its group** to get the relative frequency.

Think about using transform() after grouping — it lets you create a series aligned with the original dataframe, so you can do element-wise division directly.

Try writing a snippet where you:

* group by 'date' (or the group column)
* calculate the sum of 'count-sum' within each group using .transform('sum')
* then divide 'count-sum' by this transformed sum to get relative frequency per group

If you want, I can help you refine that once you try! Does that make sense?

**You said:**

i don't understand the tranmsform('sum') so i am trying a code here tell me if I am doing it right? month\_total = df.groupby('date')['count-sum'].sum().reset\_index(name='total\_per\_month')

**ChatGPT said:**

You're **very close** — and you're doing a good job!

Your code:

python

Copy code

month\_total = df.groupby('date')['count-sum'].sum().reset\_index(name='total\_per\_month')

is **correct** if you're trying to calculate the **total count per date** and store it in a new DataFrame called month\_total. Then merging it back (like you did earlier) is one way to get the total per row and compute relative frequency.

**But — here’s the catch:**

If you want to **avoid merging** and do it directly on the original df, that's where .transform('sum') becomes useful.

Here’s the concept:

* groupby('date')['count-sum'].transform('sum') gives you a **column** where **each row contains the total count for its group ('date')**.
* You can then divide the original 'count-sum' by that result.

So something like this:

python

Copy code

df['relative\_freq'] = df['count-sum'] / df.groupby('date')['count-sum'].transform('sum')

This avoids the need for merging.

Would you like to compare both versions side by side?