**Student:**  
Hi, I’m a DH student working on Mini Project 3. I’m analyzing a unigram dataset and trying to filter out violence-related terms like "kill", "attack", and "bomb". I don’t need the code just yet — I want to understand the logic behind how I could approach this using pandas. Could you guide me?

**ChatGPT:**  
Of course! Great to hear you're thinking about the *why* before the *how*. Since you have a list of violence-related keywords, you'll want to check each row in your '1-gram' column to see if it contains any of those keywords. This is commonly done with string matching or regular expressions. Have you used .str.contains() in pandas before?

**Student:**  
I’ve seen .str.contains() used in examples, but I’m not exactly sure what makes it different from something like ==. Why use .str.contains() instead of just checking if a word equals "kill"?

**ChatGPT:**  
Good question! When you use ==, you're checking for *exact matches* — like "kill" and nothing else. But .str.contains() is more flexible; it lets you search for a *substring* within a string. So "killing", "kills", or "killer" would all match "kill" using .str.contains("kill"). This is helpful when you're interested in capturing variations of a word.

**Student:**  
Ah, I see. But doesn’t that also mean it might accidentally match words I don’t want — like “skill” or “killjoy”? How do I prevent those kinds of false positives?

**ChatGPT:**  
Exactly — that’s one of the trade-offs. If you’re concerned about unintended matches, you can use **regular expressions** with word boundaries. For example, using \bkill\b ensures that only standalone instances of "kill" are matched. It won’t match "skill" or "killer". In pandas, you’d pass this as a pattern inside .str.contains() with regex=True.

**Student:**  
That makes sense. What does case=False do in this context? I often see that in examples.

**ChatGPT:**  
Great catch. case=False makes the match *case-insensitive*. So it doesn’t matter whether the word appears as "Kill", "KILL", or "kill" — all will be treated the same. That’s usually useful when analyzing real-world text where capitalization can vary.

**Student:**  
Cool. One more thing — I saw some code that used na=False inside .str.contains(). Why is that necessary?

**ChatGPT:**  
Nice observation! That’s there to handle any missing values (NaNs) in the column. If a row has a NaN in '1-gram', .str.contains() would return an error unless you specify na=False, which just tells pandas to treat missing values as non-matches.

**Student:**  
Okay, now I’m getting the full picture! So, to summarize: I define my keyword list, then use .str.contains() with | to combine them, and I can add word boundaries for precision. Then I can use case=False and na=False to make it robust.

**ChatGPT:**  
Perfect summary! You're definitely understanding the logic behind the code now. Once you filter the terms, you can also group by year or tag each row with the matching keyword if you want to analyze trends. Let me know if you'd like to go further with grouping or visualizing this data.

**Conversation 2;**

**Student:**  
Hi, I'm a Digital Humanities student working on Mini Project 3. I'm analyzing unigrams over time and looking into violence-related terms like "kill", "bomb", etc. I want to highlight which keyword each unigram belongs to. Conceptually, how can I detect which specific keyword a unigram is related to without just using many if statements?

**ChatGPT:**  
That's a good approach! Instead of writing repetitive if checks for each keyword, you can loop through a list of keywords and check if any of them appear in the unigram. This way, the logic becomes much cleaner and easier to update if you want to add more keywords later. You’d typically write a function that checks each keyword against the unigram text and returns the one that matches.

**Student:**  
I see. So if a unigram like "killing" appears, and I have "kill" in my keyword list, it should return "kill"? But what if multiple keywords are present in one unigram — like "bombstrike"? How do I ensure the most appropriate match is selected?

**ChatGPT:**  
Great question! In cases like that, the function usually returns the **first match** it encounters in the list. If there's a risk of multiple matches, you could improve the logic by either:

* Prioritizing keywords (e.g., more specific ones first),
* Or using regular expressions with word boundaries to reduce false matches (like avoiding "kill" matching "skill").

**Student:**  
That makes sense. Also, once I match the keyword to a unigram, how do I store this info? Is it best to create a new column in the DataFrame for the matched keyword?

**ChatGPT:**  
Exactly — creating a new column like keyword helps keep things organized. It also makes it easier to group and visualize the data later (like in a stacked bar chart or a line graph showing trends by keyword). You can use the .apply() method with your keyword-matching function to assign values to that new column.

**Student:**  
Cool, I feel I understand the logic much better now! Thanks — I’ll try implementing this with .apply() and my violence-related keywords.