Code 1:  
I am an undergraduate Digital Humanities student working on a topic modeling project. My dataset is stored in a CSV file with multiple topic-related columns. I need to filter out unclassified articles that are labeled as "Topic -1". I want to do this using Pandas in Python. How can I efficiently filter out these rows?

Filtering Out Unclassified Articles in Pandas

To filter out rows labeled as "Topic -1" in your CSV file using Pandas, you can use the following approach:

python

Copy

Download

import pandas as pd

*# Load your CSV file*

df = pd.read\_csv('your\_file.csv')

*# Filter out rows where any topic column equals "Topic -1"*

*# Method 1: If you know the exact column name(s)*

filtered\_df = df[df['your\_topic\_column'] != "Topic -1"]

*# Method 2: If you want to check multiple topic columns*

*# First, identify which columns contain topic information (adjust as needed)*

topic\_columns = [col for col in df.columns if 'Topic' in col]

*# Then filter rows where none of the topic columns equal "Topic -1"*

filtered\_df = df[~df[topic\_columns].eq("Topic -1").any(axis=1)]

*# Method 3: If "Topic -1" appears in any column (more aggressive filtering)*

*# filtered\_df = df[~df.eq("Topic -1").any(axis=1)]*

*# Save the filtered data if needed*

filtered\_df.to\_csv('filtered\_file.csv', index=False)

Explanation:

1. **Method 1** is best when you know exactly which column contains the topic classification.
2. **Method 2** checks multiple columns that contain "Topic" in their name.
3. The ~ operator negates the condition (selects rows that DON'T match).
4. any(axis=1) checks if any value in the row matches the condition.

Alternative if "-1" appears without "Topic" prefix:

python

Copy

Download

filtered\_df = df[df['your\_topic\_column'] != -1] *# if stored as number*

Would you like me to adapt this solution for any specific structure of your CSV file?

how the method 1, can you please brief me about it

Understanding Method 1 for Filtering "Topic -1" Rows

Method 1 is the simplest and most straightforward approach when you know exactly which column contains your topic classifications. Here's a detailed breakdown:

The Basic Syntax

python

Copy

Download

filtered\_df = df[df['your\_topic\_column'] != "Topic -1"]

How It Works

1. **Column Selection**: df['your\_topic\_column'] selects the specific column containing your topic labels.
2. **Comparison Operation**: != "Topic -1" creates a Boolean mask (a Series of True/False values) where:
   * True means the value is NOT "Topic -1" (keep this row)
   * False means the value IS "Topic -1" (exclude this row)
3. **Filtering**: The DataFrame df[] uses this Boolean mask to select only rows where the condition is True.

Practical Example

Let's say your CSV has:

* A column named "Primary\_Topic" containing values like "Topic 1", "Topic 2", "Topic -1", etc.

python

Copy

Download

import pandas as pd

*# Sample data*

data = {

'Article\_ID': [1, 2, 3, 4],

'Title': ['A', 'B', 'C', 'D'],

'Primary\_Topic': ['Topic 1', 'Topic -1', 'Topic 2', 'Topic -1']

}

df = pd.DataFrame(data)

*# Filtering out Topic -1*

filtered = df[df['Primary\_Topic'] != "Topic -1"]

print(filtered)

Output:

Copy

Download

Article\_ID Title Primary\_Topic

0 1 A Topic 1

2 3 C Topic 2

Key Advantages

1. **Simplicity**: Just one line of clear, readable code
2. **Efficiency**: Pandas optimizes this operation internally
3. **Precision**: Works perfectly when you know the exact column to check

When to Use Method 1

* When you have a single column containing all topic classifications
* When the column name is known and consistent
* When you only need to filter based on one specific column

Would you like me to show how this would work with your actual column name?

Code 2:

ChatGPT

**Saved memory full**

**You said:**

I am working on a topic modeling project in Python using Pandas. I need to filter out rows from my dataframe where all topic keywords are stopwords. Here are my specific requirements: My dataframe has four columns containing topic keywords: topic\_1, topic\_2, topic\_3, and topic\_4 I have a predefined set of nltk stopwords stored in a variable called stop\_words i have checked none of the keyword columns contain null values I want to remove rows where ALL FOUR keywords are in the stopwords list Please provide: The complete Python/Pandas code to perform this filtering A clear explanation of how the code works the expected output format the solution should use this approach: Create a function to check if all words are stopwords Apply this function to filter the DataFrame update the dataframe using ~ operator

**3/3**

**ChatGPT said:**

Sure! Here's a complete solution for your problem using Python and Pandas.

**Python code:**

python

Copy code

import pandas as pd from nltk.corpus import stopwords *# Example setup (replace with your actual DataFrame and stop\_words)* stop\_words = set(stopwords.words('english')) *# Sample DataFrame for demonstration* data = { 'topic\_1': ['the', 'cat', 'and', 'data'], 'topic\_2': ['is', 'dog', 'in', 'analysis'], 'topic\_3': ['on', 'mouse', 'for', 'python'], 'topic\_4': ['at', 'lion', 'but', 'code'] } df = pd.DataFrame(data) *# Function to check if all keywords in a row are stopwords* def all\_stopwords(row): return all(word in stop\_words for word in [row['topic\_1'], row['topic\_2'], row['topic\_3'], row['topic\_4']]) *# Filter the DataFrame using the ~ operator to keep rows where NOT all words are stopwords* df\_filtered = df[~df.apply(all\_stopwords, axis=1)] print(df\_filtered)

**Explanation:**

1. **Stopwords Set**: You have a predefined set stop\_words containing stopwords (like 'the', 'is', 'on', etc.).
2. **Function**all\_stopwords: This function takes a row of the dataframe, extracts the four topic keyword columns, and checks if *all* of those keywords are present in the stopwords set.
   * It returns True if all four are stopwords, False otherwise.
3. **Filtering**: We use df.apply(all\_stopwords, axis=1) which applies this function row-wise across the DataFrame and returns a boolean Series.
4. **Using**~**(NOT operator)**: We invert the boolean Series to select rows where **not all** keywords are stopwords, i.e., keep rows where at least one keyword is **not** a stopword.
5. **Update DataFrame**: Finally, we filter the DataFrame by selecting only those rows, resulting in df\_filtered.

**Expected output format:**

The filtered DataFrame will have the same columns (topic\_1, topic\_2, topic\_3, topic\_4), but **rows where all four keywords are stopwords will be removed**.

For the sample data above, output would be:

bash

Copy code

topic\_1 topic\_2 topic\_3 topic\_4

1 cat dog mouse lion

3 data analysis python code

Here, the first and third rows from the original DataFrame were removed because all their topic keywords were stopwords.

If you want me to help with your actual DataFrame or have more questions, just let me know!

**You said:**

def all\\_stopwords(row): return all(word in stop\\_words for word in \[row\['topic\\_1'\], row\['topic\\_2'\], row\['topic\\_3'\], row\['topic\\_4'\]\])

how this code works

**ChatGPT said:**

Got it! Here's a detailed, step-by-step explanation of **how this specific function works**:

python

Copy code

def all\_stopwords(row): return all(word in stop\_words for word in [row['topic\_1'], row['topic\_2'], row['topic\_3'], row['topic\_4']])

**Step-by-step breakdown:**

1. **Function Input:**
   * The function all\_stopwords takes a single argument called row.
   * When you use df.apply(all\_stopwords, axis=1), each row is a single row from your DataFrame, represented as a Pandas Series.
   * For example, a row might look like:

vbnet

Copy code

topic\_1 "the"

topic\_2 "is"

topic\_3 "on"

topic\_4 "at"

Name: 0, dtype: object

1. **Extract keywords from the row:**
   * The expression [row['topic\_1'], row['topic\_2'], row['topic\_3'], row['topic\_4']] creates a Python list containing the values from the four topic keyword columns in that row.
   * Example: ["the", "is", "on", "at"]
2. **Check if each word is a stopword:**
   * word in stop\_words checks whether a single word is present in the predefined stopwords set stop\_words.
   * For example, "the" in stop\_words would return True if "the" is a stopword.
3. **Generator expression inside**all()**:**
   * The expression

python

Copy code

(word in stop\_words for word in [...])

is a **generator expression**. It iterates over each word in the list and produces a series of boolean values:

* + - True if the word is in stop\_words
    - False otherwise
  + For example, for ["the", "is", "on", "at"] it might produce: True, True, True, True

1. **Using**all()**:**
   * The built-in Python function all() takes an iterable of boolean values and returns:
     + True if **every single item** in the iterable is True
     + False if **any** item is False
   * So all(...) here means:  
     "Are **all** of the four topic keywords stopwords?"
2. **Return value:**
   * The function returns True if all four topic keywords are stopwords, otherwise it returns False.

**Summary:**

* This function checks **if all four keywords in a given row are stopwords**.
* It extracts the four keywords from the row.
* Checks each keyword against the stopwords list.
* Returns True only if **every keyword** is a stopword.
* This boolean return value is used to filter out those rows from your DataFrame.

If you want, I can also help explain how this function interacts with the DataFrame filtering line!

Code 3

**You said:**

now that I have a dataframe in Python with columns: Topic, topic\_1, topic\_2, topic\_3, and topic\_4. i want to manually review the topics. how can i print the topics, so that i can review them later

**ChatGPT said:**

Great question! If you want to manually review the topics in your DataFrame, a good way is to **group by each Topic number**, get the **first set of keywords** for that topic, and **print them in a readable format**.

**✅ Here’s the code you can use:**

python

Copy code

# Get the first set of keywords for each topic

topic\_keywords = df.groupby("Topic")[["topic\_1", "topic\_2", "topic\_3", "topic\_4"]].first().reset\_index()

# Print topics and their keywords

for \_, row in topic\_keywords.iterrows():

topic\_num = row["Topic"]

words = ", ".join([row["topic\_1"], row["topic\_2"], row["topic\_3"], row["topic\_4"]])

print(f"Topic {topic\_num}: {words}")

**🔍 Explanation:**

1. **groupby("Topic")**:  
   Groups your DataFrame by the Topic column, so you can handle each topic separately.
2. **.first()**:  
   Gets the **first row** from each topic group—usually enough for a quick review.
3. **.reset\_index()**:  
   Turns the Topic back into a regular column so it’s easy to loop over.
4. **Loop and print()**:  
   For each topic group, the code joins the four keywords into a single string using ", ".join() and prints them like:

yaml

Copy code

Topic 0: climate, change, temperature, carbon

Topic 1: economy, inflation, trade, growth

This makes it easy to scan through all your topics and **manually assess their quality or meaning**. You can even redirect the output to a text file later if you want to keep a record.

**You said:**

how the loop works here

**ChatGPT said:**

Great question! Let’s break down **how the loop works** in this code:

**🔁 The Loop Code:**

python

Copy code

for \_, row in topic\_keywords.iterrows():

topic\_num = row["Topic"]

words = ", ".join([row["topic\_1"], row["topic\_2"], row["topic\_3"], row["topic\_4"]])

print(f"Topic {topic\_num}: {words}")

**🧠 What’s happening step by step:**

1. **for \_, row in topic\_keywords.iterrows():**
   * This loop goes **row by row** through the topic\_keywords DataFrame (which contains one row per topic).
   * iterrows() returns a pair:
     + The first part (here \_) is the index of the row — we don’t need it, so we use \_.
     + The second part (row) is a **Pandas Series** representing one row (i.e. one topic and its 4 keywords).
2. **topic\_num = row["Topic"]**
   * Gets the topic number from the Topic column of that row.
3. **[row["topic\_1"], row["topic\_2"], row["topic\_3"], row["topic\_4"]]**
   * Grabs the four keyword values from the current row.
4. **", ".join([...])**
   * Joins the four keywords into a single string separated by commas (e.g. "health, care, hospital, disease").
5. **print(f"Topic {topic\_num}: {words}")**
   * Prints the topic number and its keywords in a clean, readable format.

**✅ Example Output:**

Let’s say one row has:

python

Copy code

row["Topic"] = 2

row["topic\_1"] = "health"

row["topic\_2"] = "care"

row["topic\_3"] = "hospital"

row["topic\_4"] = "disease"

The output would be:

yaml

Copy code

Topic 2: health, care, hospital, disease

This loop helps you **inspect all topics manually**, one by one. Let me know if you’d like to also save this output to a file!