**ChatGPT Conversation 1**

i am a student of Digital humanities and i am working on a project where i have to produce a distant reading analysis of Al-Jazeera Gaza corpus. i am using a dataset of tfidf(similarity) . my approach is to know that do the articles that have high tf-IDF similarity also belong to the same topic cluster?. for that i am trying to merge the tf-idf dataset with topic-model data set. the tf-idf dataset has following columns : filename-1,filename-2,similarity,title-1,year-1,month-1,day-1,title-2,year-2,month-2,day-2. and the topic-model dataset has following columns: year, month, day, title, file, topic, count, topic-1, topic-2, topic-3, topic-4. how do i merge them? for your convinience i will share my script also. import pandas as pd #read the csv file df = pd.read\_csv("C:\\Users\\HP\\Downloads\\FASDH25-portfolio3\\data\\dataframes\\tfidf\\tfidf-over-0.3.csv") #filter out rows where similarity score is greater than equa to 0.7 high\_similarity = df[df["similarity"] >= 0.7] #save the high\_similarity rows to csv high\_similarity.to\_csv("C:\\Users\\HP\\Downloads\\FASDH25-portfolio3\\data\\dataframes\\tfidf\\tfidf-over-0.7.csv", index=False) #print first few rows to confirm print(high\_similarity.head()) #read the topic model csv file topic\_df = pd.read\_csv("C:\\Users\\HP\\Downloads\\FASDH25-portfolio3\\data\\dataframes\\topic-model\\topic-model.csv") #print first few rows of topic model data print(topic\_df.head())

**ChatGPT said:**

Great. You're asking the right kind of analytical question: **"Do articles that have high TF-IDF similarity also share the same topic cluster?"** To answer this, you need to **merge** your high\_similarity DataFrame with the topic\_df **twice**: once for filename-1 and once for filename-2. Here's how you can do it properly and cleanly:

**✅ Step-by-step Code to Merge**

python

CopyEdit

import pandas as pd

# Read TF-IDF similarity data

df = pd.read\_csv("C:\\Users\\HP\\Downloads\\FASDH25-portfolio3\\data\\dataframes\\tfidf\\tfidf-over-0.3.csv")

# Filter similarity >= 0.7

high\_similarity = df[df["similarity"] >= 0.7]

# Save filtered data (optional)

high\_similarity.to\_csv("C:\\Users\\HP\\Downloads\\FASDH25-portfolio3\\data\\dataframes\\tfidf\\tfidf-over-0.7.csv", index=False)

# Read topic model data

topic\_df = pd.read\_csv("C:\\Users\\HP\\Downloads\\FASDH25-portfolio3\\data\\dataframes\\topic-model\\topic-model.csv")

# Merge with topic model data for filename-1

merged1 = pd.merge(

high\_similarity,

topic\_df,

how="left",

left\_on=["filename-1"],

right\_on=["file"]

)

# Rename topic columns to avoid confusion

merged1 = merged1.rename(columns={

"topic": "topic\_1",

"topic-1": "topic\_1\_1",

"topic-2": "topic\_1\_2",

"topic-3": "topic\_1\_3",

"topic-4": "topic\_1\_4"

})

# Merge with topic model data for filename-2

merged2 = pd.merge(

merged1,

topic\_df,

how="left",

left\_on=["filename-2"],

right\_on=["file"]

)

# Rename again to distinguish second article's topics

merged2 = merged2.rename(columns={

"topic": "topic\_2",

"topic-1": "topic\_2\_1",

"topic-2": "topic\_2\_2",

"topic-3": "topic\_2\_3",

"topic-4": "topic\_2\_4"

})

# Drop redundant 'file' columns

merged2 = merged2.drop(columns=["file\_x", "file\_y"])

# Save final merged DataFrame

merged2.to\_csv("C:\\Users\\HP\\Downloads\\FASDH25-portfolio3\\data\\dataframes\\merged\\high-similarity-with-topics.csv", index=False)

# Print to check

print(merged2.head())

**🧠 What this Script Does**

* Merges **high TF-IDF similar article pairs** with their **respective topic model data**.
* Clearly distinguishes the topic clusters of filename-1 and filename-2.
* Allows you to **compare if articles in a highly similar pair share any topic clusters.**

**✅ Next Analytical Step**

After merging, you can write logic to **check topic overlap**:

python

CopyEdit

def has\_topic\_overlap(row):

topics\_1 = {row['topic\_1\_1'], row['topic\_1\_2'], row['topic\_1\_3'], row['topic\_1\_4']}

topics\_2 = {row['topic\_2\_1'], row['topic\_2\_2'], row['topic\_2\_3'], row['topic\_2\_4']}

return len(topics\_1.intersection(topics\_2)) > 0

# Add a column indicating if there is any topic overlap

merged2['shared\_topic'] = merged2.apply(has\_topic\_overlap, axis=1)

# How many high-similarity article pairs share topics?

print(merged2['shared\_topic'].value\_counts())

**🧪 Result**

This will help you **quantify** how often high similarity implies **topical clustering** — key for your distant reading insight.

Let me know if you’d like help visualizing these results too (bar chart, pie, etc).

**You said:**

i have done it in this way # Merge filename-1 with topic words merged\_df = tfidf\_df.merge(topics\_df, how='left', left\_on='filename-1', right\_on='file') merged\_df = merged\_df.rename(columns={ 'topic\_1': 'topic1\_file1', 'topic\_2': 'topic2\_file1', 'topic\_3': 'topic3\_file1', 'topic\_4': 'topic4\_file1' }) merged\_df = merged\_df.drop(columns=['file', 'year', 'month', 'day']) # Merge filename-2 with topic words merged\_df = merged\_df.merge(topics\_df, how='left', left\_on='filename-2', right\_on='file') merged\_df = merged\_df.rename(columns={ 'topic\_1': 'topic1\_file2', 'topic\_2': 'topic2\_file2', 'topic\_3': 'topic3\_file2', 'topic\_4': 'topic4\_file2' }) merged\_df = merged\_df.drop(columns=['file', 'year', 'month', 'day']) # Save the merged dataframe merged\_df.to\_csv("C:/Users/HP/Downloads/FASDH25-portfolio3/data/dataframes/tfidf/tfidf-over-0.7-with-topics.csv", index=False) # Print preview print(merged\_df.head()) is it right?

**ChatGPT said:**

You're very close — just a **small but critical issue** in your code:

**❌ Problem:**

You're renaming the topic columns like this:

python

CopyEdit

'\*\*topic\_1\*\*': 'topic1\_file1',

'\*\*topic\_2\*\*': 'topic2\_file1',

But in your **original topics\_df**, the columns are:

plaintext

CopyEdit

topic, topic-1, topic-2, topic-3, topic-4

That means you're trying to rename columns that **don’t exist** — topic\_1, topic\_2, etc. So your renamed columns are either missing or filled with NaNs silently.

**✅ What You Should Fix**

Update your renaming to **match the actual column names**:

python

CopyEdit

merged\_df = merged\_df.rename(columns={

'topic-1': 'topic1\_file1',

'topic-2': 'topic2\_file1',

'topic-3': 'topic3\_file1',

'topic-4': 'topic4\_file1'

})

Same for the second merge:

python

CopyEdit

merged\_df = merged\_df.rename(columns={

'topic-1': 'topic1\_file2',

'topic-2': 'topic2\_file2',

'topic-3': 'topic3\_file2',

'topic-4': 'topic4\_file2'

})

**🔁 Full Corrected Version:**

python

CopyEdit

# Merge filename-1 with topic words

merged\_df = tfidf\_df.merge(topics\_df, how='left', left\_on='filename-1', right\_on='file')

merged\_df = merged\_df.rename(columns={

'topic-1': 'topic1\_file1',

'topic-2': 'topic2\_file1',

'topic-3': 'topic3\_file1',

'topic-4': 'topic4\_file1'

})

merged\_df = merged\_df.drop(columns=['file', 'year', 'month', 'day'])

# Merge filename-2 with topic words

merged\_df = merged\_df.merge(topics\_df, how='left', left\_on='filename-2', right\_on='file')

merged\_df = merged\_df.rename(columns={

'topic-1': 'topic1\_file2',

'topic-2': 'topic2\_file2',

'topic-3': 'topic3\_file2',

'topic-4': 'topic4\_file2'

})

merged\_df = merged\_df.drop(columns=['file', 'year', 'month', 'day'])

# Save the merged dataframe

merged\_df.to\_csv("C:/Users/HP/Downloads/FASDH25-portfolio3/data/dataframes/tfidf/tfidf-over-0.7-with-topics.csv", index=False)

# Print preview

print(merged\_df.head())

**💡 Tip for Next Step:**

Now that you’ve merged correctly, you can check **topic overlap**:

python

CopyEdit

def has\_overlap(row):

topics\_file1 = {row['topic1\_file1'], row['topic2\_file1'], row['topic3\_file1'], row['topic4\_file1']}

topics\_file2 = {row['topic1\_file2'], row['topic2\_file2'], row['topic3\_file2'], row['topic4\_file2']}

return len(topics\_file1 & topics\_file2) > 0

merged\_df['topic\_overlap'] = merged\_df.apply(has\_overlap, axis=1)

Then analyze or visualize how often high-similarity articles share topic clusters.

Let me know when you’re ready for plotting or writing this up in your report.

**ChatGPT Conversation 2**

i want to bring all the columns that relate tofilename 2 under the columns of filename 1 and concatenate

**ChatGPT said:**

Great! If you want to **combine the topic-related columns from both filename-1 and filename-2** into one **stacked or concatenated format**, you likely want to reshape the data so that **each article in a pair becomes its own row** — instead of having both in the same row.

This is useful if you're analyzing or visualizing **individual articles** rather than pairs.

**✅ Goal:**

Turn this:

CopyEdit

| filename-1 | topic1\_file1 | ... | filename-2 | topic1\_file2 | ...

Into this:

CopyEdit

| filename | topic1 | topic2 | ...

| filename | topic1 | topic2 | ...

**✅ Steps to do that:**

You already have:

* merged\_df with topic info for filename-1 and filename-2

Now:

python

CopyEdit

# First: select and rename columns related to filename-1

file1\_df = merged\_df[[

'filename-1', 'main\_topic\_file1', 'topic1\_file1', 'topic2\_file1', 'topic3\_file1', 'topic4\_file1'

]].rename(columns={

'filename-1': 'filename',

'main\_topic\_file1': 'main\_topic',

'topic1\_file1': 'topic1',

'topic2\_file1': 'topic2',

'topic3\_file1': 'topic3',

'topic4\_file1': 'topic4'

})

# Then: do the same for filename-2

file2\_df = merged\_df[[

'filename-2', 'main\_topic\_file2', 'topic1\_file2', 'topic2\_file2', 'topic3\_file2', 'topic4\_file2'

]].rename(columns={

'filename-2': 'filename',

'main\_topic\_file2': 'main\_topic',

'topic1\_file2': 'topic1',

'topic2\_file2': 'topic2',

'topic3\_file2': 'topic3',

'topic4\_file2': 'topic4'

})

# Finally: concatenate both

stacked\_df = pd.concat([file1\_df, file2\_df], ignore\_index=True).drop\_duplicates()

# Optional: sort if needed

stacked\_df = stacked\_df.sort\_values(by='filename')

# Print or save

print(stacked\_df.head())

**✅ Result:**

stacked\_df will now have **one row per article** with all its topic data — regardless of whether it appeared in filename-1 or filename-2.

Let me know if you'd like to group by main\_topic or visualize the distribution next!

**ChatGPT Conversation 3:**

my bars are very light in color which is making it difficult to read. How I make the colours bold?

**ChatGPT said:**

To make your bars darker and more readable, you need to customize the color palette used by Plotly. Here's how you can do it **:**

Modify the px.bar() section by **adding a custom color\_discrete\_sequence**. Plotly has built-in color sequences like "Plotly", "Viridis", "Dark2", etc. These produce clearer, more contrasted bars.

* Add color\_discrete\_sequence=px.colors.qualitative.Dark2 ensures each bar is assigned a **darker, contrasting color**.
* This improves **visual clarity** and **accessibility**, especially when printing or showing the graph on projectors/screens.

Let me know if you’d prefer a different color palette like "Bold" or "Set1"

Top of Form

**Conversation 0:**

in my merged csv,, there is a column "TopicNumer" and in some cases there is -1TopicNumber which contain words like "the, to, of, me" i want to remove them. How do I do that?Bottom of Form

**ChatGPT said:**

To **remove rows where TopicNumber is -1**, which represent stopword-like topics (e.g., containing only "the, to, of"), you just need to **filter out those rows** before doing the grouping and plotting.

df = df[df["TopicNumber"] != -1]

This ensures that the "-1" topic (which usually corresponds to stopwords or junk clusters) is **completely removed** from your bar chart and from the final HTML visualization. Let me know if you want to remove these rows earlier in the pipeline or filter based on specific topic labels too.

**Conversation 4:**

i have a dataset and i want to check whether there are any empty or missing values in my dataset or not. act as a good tutor and make me understand

2/2

**ChatGPT said:**

Great — you're asking a very important and foundational question in data analysis. Detecting **missing or empty values** is one of the first things you should always do when you begin working with a dataset. Let me walk you through it step by step, like a good tutor should.

**🧠 Step 1: Understand What "Missing" Means in Pandas**

In Python (using the pandas library), **missing values** are usually represented by:

* NaN (Not a Number)
* None (Python's null value)  
  These are treated as *nulls*, and pandas has special tools to deal with them.

**🧪 Step 2: Use .isnull() and .sum() Together**

Here’s the simplest and most clear method to check for missing values in **each column** of your dataset:

python

CopyEdit

import pandas as pd

# load your dataset, e.g.

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

# check for missing values

print(df.isnull().sum())

**What this does:**

* df.isnull() returns a DataFrame of the same shape with True for missing values and False otherwise.
* .sum() adds up the True values in each column, giving you a total count of missing entries.

**🧮 Example:**

Let’s say your dataset looks like this:

| **name** | **age** | **city** |
| --- | --- | --- |
| Alice | 25 | Karachi |
| Bob |  | Lahore |
| Charlie | 30 |  |

Running the code:

python

CopyEdit

df.isnull().sum()

Would output:

go

CopyEdit

name 0

age 1

city 1

dtype: int64

Which means:

* No missing values in name
* 1 missing value in age
* 1 missing value in city

**🔍 Extra Tip: Check If There Are *Any* Missing Values At All**

You can do:

python

CopyEdit

print(df.isnull().values.any())

This will return:

* True if there are any missing values anywhere in the dataset
* False if the dataset is completely clean

**🚧 Optional: Also Check for Empty Strings ("")**

Sometimes missing data is stored as empty strings ("") instead of real nulls. To catch those:

python

CopyEdit

print((df == "").sum())

This checks for *empty strings*, not just NaN. If you find some, you can convert them to real missing values using:

python

CopyEdit

df.replace("", pd.NA, inplace=True)

**✅ Summary of What You Should Do First**

1. Use df.isnull().sum() to see null counts per column.
2. Use df.isnull().values.any() to check if there are **any** nulls.
3. Optionally, check for "" empty strings too.
4. (Only if needed) Replace those empty strings with pd.NA.

Would you like to try this on your dataset now? If yes, paste your code or tell me what you're stuck with, and I’ll guide you further.

Top of Form

Bottom of Form