I need help in my project 3 of digital humanities course. r u ready

**ChatGPT said:**

Absolutely, I'm ready to help! Just let me know the details of your Project 3—what it's about, any guidelines or goals, and what specific assistance you need.

**You said:**

so i am using an n-gram dataset. I want to compare fear-related and hope-related words that are in the corpus and i am working on my script for visualisation graph. i have imported pandas and defined the path. i have also created the date columns for filtering the data. now i want to add a new column called category that labels each 1-gram as either fear or hope. how should i do that. dont give me the code first. explain

**ChatGPT said:**

Great, you're on a solid track!

To add a new column called **category** that labels each 1-gram as either **fear** or **hope**, here's how to think about the process step by step:

**✅ Step-by-Step Breakdown**

**1. Define Your Word Lists**

You need two sets (or lists) of words:

* One containing words that express **fear** (e.g., *terror, panic, danger*).
* One containing words that express **hope** (e.g., *dream, future, believe*).

These will act like dictionaries that your script will use to look up and tag each word.

**2. Check Each Row's 1-Gram**

Your dataset should have a column (say, word or ngram) where each row is a 1-gram. For each of these:

* If the word is in the fear list, label the row as "fear".
* If it’s in the hope list, label it "hope".
* If it's in neither, you can choose to:
  + Leave it blank (NaN)
  + Or assign a third label like "none" or "other"—depending on your goals.

**3. Create the New Column**

Using some conditional logic, go row by row (or use a vectorized method for speed) and assign the appropriate category value.

**4. Case Sensitivity and Cleaning**

Make sure your matching is **case-insensitive**, and the words are cleaned (e.g., no punctuation or whitespace issues), or else the matching could fail.

**📌 Example Considerations**

* Are you only labeling words that are explicitly in your fear/hope lists?
* Will you do any stemming or lemmatization (e.g., *fearing* → *fear*)?
* How will you handle ambiguous words that could mean both (e.g., *anxious*)?

Let me know if that makes sense or if you want to walk through how to write the actual code for that part.

**You said:**

so basically i am applying a function to each 1-gram that checks if it is in either list and labels it accordingly. I want to group the data later by emotional category. show me the code

**ChatGPT said:**

Exactly right—you’re applying a function to each 1-gram, checking if it's in either the **fear** or **hope** list, and labeling it. Here's how you can do that in code using pandas:

**✅ Step-by-Step Python Code**

python

CopyEdit

import pandas as pd

# Example: Load your dataset

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

# Define your fear- and hope-related word lists

fear\_words = {"terror", "panic", "danger", "anxiety", "fear", "dread"}

hope\_words = {"hope", "dream", "believe", "future", "faith", "aspire"}

# Ensure the 1-gram column (e.g., "ngram") is lowercase for matching

df["ngram"] = df["ngram"].str.lower()

# Define a function to assign categories

def assign\_category(word):

if word in fear\_words:

return "fear"

elif word in hope\_words:

return "hope"

else:

return "none"

# Apply the function to create a new column

df["category"] = df["ngram"].apply(assign\_category)

# Now you can group by category

grouped = df.groupby("category").sum(numeric\_only=True) # or use count(), depending on what you want

# Optional: view the result

print(grouped)

**🛠 Notes:**

* Replace "ngram" with the actual name of your 1-gram column.
* You can tweak "none" to pd.NA or exclude those rows later if you want.
* Make sure the column you group by is actually meaningful to your visualization—this will feed into your graph.

Let me know if you want to move on to visualizing this with matplotlib, seaborn, or another library.

**You said:**

not the whole code i have done it just the catagorization part

**ChatGPT said:**

Perfect! Here's **just the categorization part** (the essential logic to create the category column):

python

CopyEdit

# Define your word sets

fear\_words = {"terror", "panic", "danger", "anxiety", "fear", "dread"}

hope\_words = {"hope", "dream", "believe", "future", "faith", "aspire"}

# Make sure the ngram column is lowercase

df["ngram"] = df["ngram"].str.lower()

# Categorize each 1-gram

def assign\_category(word):

if word in fear\_words:

return "fear"

elif word in hope\_words:

return "hope"

else:

return "none"

# Apply the function

df["category"] = df["ngram"].apply(assign\_category)

This will label each row in your DataFrame as "fear", "hope", or "none" based on the word in the ngram column. Let me know if you want to exclude "none" rows or add more nuance (like stemming or scoring).

**You said:**

ok so df['category'] = df['1-gram'].apply( lambda word: 'fear' if word in Fear\_related\_filter\_list else 'hope'

**ChatGPT said:**

You're almost there! But the code you're writing will **label all non-fear words as "hope"**, which might not be what you want—especially if a word is in **neither** list.

To handle all three cases (fear, hope, none), you need to explicitly check both lists.

Here's a corrected version of your lambda function:

python

CopyEdit

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

lambda word: 'fear' if word in Fear\_related\_filter\_list

else ('hope' if word in Hope\_related\_filter\_list

else 'none')

)

**Why this works:**

* It first checks if the word is in the **fear list** → label as "fear".
* If not, it checks if it's in the **hope list** → label as "hope".
* If it's in **neither**, it falls back to "none".

Let me know if you'd prefer to drop the "none" rows right after this step.

**You said:**

i have plotted my grapph give me a code to Format bar labels to show 2 decimal points and position them clearly above each bar

**ChatGPT said:**

Nice work on getting to the visualization step! Here's how you can **format bar labels** to show **2 decimal places** and position them **clearly above each bar** in a matplotlib bar chart:

python

CopyEdit

import matplotlib.pyplot as plt

# Assuming you already have your bar chart plotted

ax = df.groupby("category")["some\_numeric\_column"].mean().plot(kind="bar", color=["red", "green"])

# Add labels above bars

for container in ax.containers:

for bar in container:

height = bar.get\_height()

ax.text(

bar.get\_x() + bar.get\_width() / 2, # x position (center of bar)

height + 0.01, # y position (just above bar)

f'{height:.2f}', # label with 2 decimal places

ha='center', va='bottom', fontsize=10

)

# Optional: tidy up the layout

plt.tight\_layout()

plt.ylabel("Your Y-axis Label")

plt.title("Your Chart Title")

plt.show()

**🔧 Customize if needed:**

* Adjust height + 0.01 to move labels higher or lower.
* Change fontsize for readability.
* Swap out .mean() with .sum() or whatever aggregation you're plotting.

Let me know what library you're using if you're not using matplotlib—I can adjust the example!

**You said:**

this is my plot # Plot using average mentions per article fig = px.bar( monthly\_means, x='month', # x-axis: Month y='count-mean', # y-axis: Avg mentions per article color='category', # Different color for fear and hope barmode='group', # Bars for fear and hope are side by side text='count-mean', # Display numeric values above each bar title='Fear vs Hope – Avg Mentions per Article (Sep 7, 2023 – Apr 16, 2024)', labels={'month': 'Month', 'count-mean': 'Avg Mentions per Article'}, color\_discrete\_map={'fear': 'red', 'hope': 'blue'}

**ChatGPT said:**

Thanks for sharing your code — since you're using **Plotly Express (px.bar)**, formatting the bar labels to **2 decimal places** and positioning them **clearly above each bar** just takes a couple of tweaks.

Here’s how you can modify your code to do that:

**✅ Add this to your plot:**

python

CopyEdit

fig.update\_traces(

texttemplate='%{text:.2f}', # Format text to 2 decimal places

textposition='outside' # Place labels clearly above the bars

)

**📌 Final Code (with label formatting):**

python

CopyEdit

fig = px.bar(

monthly\_means,

x='month',

y='count-mean',

color='category',

barmode='group',

text='count-mean',

title='Fear vs Hope – Avg Mentions per Article (Sep 7, 2023 – Apr 16, 2024)',

labels={'month': 'Month', 'count-mean': 'Avg Mentions per Article'},

color\_discrete\_map={'fear': 'red', 'hope': 'blue'}

)

# Format the bar labels

fig.update\_traces(

texttemplate='%{text:.2f}',

textposition='outside'

)

# Optional: Make layout tighter

fig.update\_layout(

uniformtext\_minsize=8,

uniformtext\_mode='hide',

yaxis\_title='Avg Mentions per Article'

)

fig.show()

Let me know if the bars are too short to show labels clearly — we can tweak the layout or scale to fix that too.

**You said:**

what does this code actually do

**ChatGPT said:**

Great question — let's break down **exactly what this code does**, especially the parts that format the **bar labels** in your Plotly bar chart:

**🧠 Your Chart (Recap)**

You're plotting a grouped bar chart comparing **fear** and **hope** word mentions per article over time (monthly\_means). Now you're adding **labels above the bars** showing the average value per article — formatted to **2 decimal places**.

**🧩 What Each Part Does**

**✅ text='count-mean'**

* Tells Plotly which column to use for bar **labels** (values shown on top of each bar).
* In your case, this is the average mentions per article.

**✅ fig.update\_traces(...)**

This modifies how the labels (texts) are **formatted and positioned**:

python

CopyEdit

fig.update\_traces(

texttemplate='%{text:.2f}', # Format numbers: 2 decimal places

textposition='outside' # Position: clearly \*above\* the bars

)

* **texttemplate='%{text:.2f}'**
  + Formats the number using Python string formatting.
  + .2f = show 2 digits after the decimal point.
  + Example: 1.356 → 1.36, 0.8 → 0.80.
* **textposition='outside'**
  + Places the label *above* each bar.
  + Other options include 'inside', 'auto', 'none'.

**🧹 fig.update\_layout(...) *(optional cleanup)***

python

CopyEdit

fig.update\_layout(

uniformtext\_minsize=8,

uniformtext\_mode='hide',

yaxis\_title='Avg Mentions per Article'

)

* **uniformtext\_minsize=8**: Ensures text isn't too small.
* **uniformtext\_mode='hide'**: Hides labels that don’t fit (prevents overlap/clutter).
* **yaxis\_title='...'**: Labels the Y-axis properly.

**✅ Summary**

Together, these updates:

* **Cleanly display** average mentions per article.
* **Format** the numbers to 2 decimal places.
* **Position** them nicely above the bars.
* **Improve readability** of your graph.

Let me know if you’d like to animate this or add interactivity like hover info or dropdown filters!

**You said:**

I have made my script. i need help in filtering part

**ChatGPT said:**

Great! Filtering is an important step — just let me know:

1. **What exactly you want to filter?**  
   (e.g., a specific date range, only "fear" words, or removing "none" category?)
2. **What does your DataFrame look like at that point?**  
   (Which columns are relevant to the filter — like date, category, count, etc.?)
3. **What format is your date column in?**  
   (e.g., datetime64[ns], string, or something else?)

Once I know that, I’ll guide you through writing a clean and efficient filtering step.

**You said:**

i want to filter the dataset to keep rows onlyy where the 1-gram appears in our defined word lists

**ChatGPT said:**

Perfect — so your goal is to keep only the rows where the 1-gram is in either your **fear** or **hope** word lists. That means you're filtering out everything not relevant to your emotional analysis.

**✅ What You Need to Do**

Since you already have two sets/lists like:

python

CopyEdit

fear\_words = {"terror", "panic", "anxiety", ...}

hope\_words = {"hope", "dream", "believe", ...}

You can **combine these into one set** and filter your DataFrame using .isin().

**✅ Filter Code**

python

CopyEdit

# Combine both word lists into one set

relevant\_words = fear\_words.union(hope\_words)

# Ensure the 1-gram column is lowercase for matching

df['1-gram'] = df['1-gram'].str.lower()

# Filter the DataFrame to keep only rows with relevant 1-grams

filtered\_df = df[df['1-gram'].isin(relevant\_words)]

**🔍 What This Does**

* **union()** merges the two sets into one — all relevant emotional words.
* **df['1-gram'].isin(...)** returns a Boolean Series — True where the word is in the list.
* **Filtering** applies that mask to keep only desired rows.

Let me know if you want to **filter by category** (e.g., just "fear") or by a **date range** at the same time — happy to show you how to combine filters too.

**You said:**

explain the code

**ChatGPT said:**

Absolutely — here’s a clear explanation of what each line in the filtering code does and **why it’s needed** for your project.

**🧾 Your Filtering Goal:**

Keep only rows where the 1-gram is in your **fear-related** or **hope-related** word lists.

**🧠 Step-by-Step Breakdown:**

python

CopyEdit

# 1. Combine both word lists into one set

relevant\_words = fear\_words.union(hope\_words)

* **fear\_words** and **hope\_words** are sets of words.
* **.union()** creates a new set that includes **all unique words** from both.
* ✅ This gives you a master list of emotionally relevant words.

python

CopyEdit

# 2. Ensure the 1-gram column is lowercase for matching

df['1-gram'] = df['1-gram'].str.lower()

* You normalize the text to **lowercase** so words like "Hope" or "PANIC" match your lists.
* 🔠 Matching is **case-sensitive** by default, so this step prevents mismatches.

python

CopyEdit

# 3. Filter the DataFrame to keep only rows with relevant 1-grams

filtered\_df = df[df['1-gram'].isin(relevant\_words)]

* **df['1-gram'].isin(relevant\_words)** creates a Boolean Series:
  + True for rows where the word is in your set.
  + False for all others.
* **df[...]** uses that Boolean mask to select only matching rows.
* ✅ You now have a new DataFrame (filtered\_df) with only the emotional words you care about.

**🧪 Example**

Say your DataFrame has these 1-grams:

| **1-gram** |
| --- |
| panic |
| love |
| dream |
| violence |

If your word lists are:

python

CopyEdit

fear\_words = {"panic", "violence"}

hope\_words = {"dream"}

Then after filtering, only these rows stay:

| **1-gram** |
| --- |
| panic |
| dream |
| violence |

✅ love is dropped because it’s not in either list.

Let me know if you want to apply this filtering to a larger pipeline — like before grouping or plotting!