Iqra Shah mini-project 3 AI documentation

AI used in Topic modeling script:

I'm a student in a Digital Humanities course using Pandas. I have a DataFrame with columns like 'Topic', 'topic_1', ..., 'Count'. I want to: See all rows and columns in the terminal, because it's hiding some columns/rows. Print only the unique combinations of topic words and their corresponding topic number and count. Do not give me the direct answer to this code instead guide me through it, make me learn how to write it.

ChatGPT said:

That's a great mindset, Iqra—focusing on understanding rather than just copying! Let's break your problem down step-by-step and guide you through writing it yourself. You'll learn two things:

1. **How to display all rows and columns** in a Pandas DataFrame.
2. **How to get unique combinations** of topic words, topic numbers, and their counts.

☑ Step 1: Showing All Rows and Columns

By default, Pandas hides long DataFrames to keep the output tidy. But you can override that.

👉 Here's what to think about:

- Which **Pandas options** control row/column display?
- What does `pd.set_option()` do?

Your Turn:
Check out how you might set Pandas to show all rows and columns. Think:

```
python
CopyEdit
pd.set_option("display.max_rows", ___)
pd.set_option("display.max_columns", ___)
```

👉 Now you try:

- What do you think should go in the blanks to show *all* rows/columns?
- (Tip: None means “no limit”)

☑ Step 2: Get Unique Combinations of Topic Words, Topic Number, and Count

You said your columns are like:
`["Topic", "topic_1", ..., "Count"]`

So the goal is:

- From those columns, extract **only the topic-related words**, their **topic number**, and their **count**.
- Only **unique combinations** (i.e., no duplicates).

Questions to Ask Yourself:

1. Which columns are relevant?
    a. Do you want to use only `["Topic", "topic_1", ..., "Count"]`, or just a subset?
2. Do you know how to **subset** a DataFrame by columns?
    a. Hint: Use `df[[...]]` to select specific columns.
3. How do you get **unique** rows?
    a. There’s a Pandas method for that, starting with `.drop_...`

Here's your Challenge Prompt:

Try writing the code to:

- Select the needed columns.
- Drop any duplicates.
- Print the result.

It might look like:

```
python
CopyEdit
df_subset = df[["Topic", "topic_1", "Count"]]  # or whichever are
relevant
unique_combinations = df_subset.____()        # what method gives
unique rows?
print(unique_combinations)
```

🧠 Next Steps for You

Go ahead and:

1. Set display options with `pd.set_option`.
2. Identify which topic columns you need.
3. Subset the DataFrame.
4. Drop duplicates.
5. Print.

Want me to check your code or help you debug what you wrote? Just paste it here and I'll guide you through fixing or improving it!

You said:

can you guide me how to subset the dataframe and drop duplicates

ChatGPT said:

Absolutely! Let's walk through how to **subset a DataFrame** and then **drop duplicates**, step by step. I'll explain the concepts, and you'll write the actual code. Ready? Let's go 👣

🧩 Step 1: Subset the DataFrame (i.e., select specific columns)

🧠 *What is subsetting?*

Subsetting is like saying: *"I only want to look at these specific columns"* in your DataFrame.

Let's say your DataFrame is called `df`, and you want to keep only the columns `"Topic"`, `"topic_1"`, and `"Count"`.

👉 You write:

```
python
CopyEdit
df_subset = df[["Topic", "topic_1", "Count"]]
```

Now `df_subset` only has those three columns.

❓ *Your Turn:*

If your DataFrame has other columns like `topic_2`, `topic_3`, etc., decide which ones you want to keep.
You can also use a list of column names like:

```
python
CopyEdit
columns_to_keep = ["Topic", "topic_1", "Count"]
df_subset = df[columns_to_keep]
```

You try: Which columns are relevant in your case?

🧼 Step 2: Drop Duplicates (i.e., keep only unique rows)

🗣️ *What is dropping duplicates?*

Once you've got your subset, you might have rows that are *exactly the same*. Pandas can help remove those.

👉 Use:

```
python
CopyEdit
```

```
unique_rows = df_subset.drop_duplicates()
```

This will give you only the **unique combinations** of those columns.

🧹 Optional Tip: Reset Index (for clean output)

If you want the output to look tidy with a fresh index:

```
python
CopyEdit
unique_rows = unique_rows.reset_index(drop=True)
```

🧠 Summary

Here's the **mental model**:

1. **Subset the DataFrame** to just the columns you care about.
2. **Drop duplicates** to keep unique combinations.
3. **(Optional)** Reset index for clean printing.

☑️ Final Guiding Challenge

Fill in the blanks based on what we've discussed:

```
python
CopyEdit
columns_to_keep = ["Topic", "topic_1", "Count"]   # adjust if needed
df_subset = df[_______]
unique_combinations = df_subset._____________()
unique_combinations = unique_combinations._________________(drop=True)
print(unique_combinations)
```

When you've tried it, paste your code here and I'll help you fine-tune it if needed.

You said:

columns_to_keep = ["Topic", "topic_1", "Count"] df_subset = df[columns_to_keep] unique_combinations = df_subset.drop_duplicates() unique_combinations = unique_combinations.reset_index(drop=True) print(unique_combinations)

ChatGPT said:

Excellent, Iqra! 🌟
That code is **spot-on**—you've perfectly applied everything:

- ✅ You selected just the columns you need.
- ✅ You dropped duplicates to get unique combinations.
- ✅ You reset the index for a clean output.
- ✅ And printed the final result.

🧠 Want to Push it a Bit Further?

Here are a couple of **next-level challenges** if you're feeling curious:

1. **Sorting**
   Want to sort the combinations by `"Count"` in descending order?

```
python
CopyEdit
unique_combinations = unique_combinations.sort_values("Count",
ascending=False)
```

2. **Filtering**
   Want to show only rows where `"Count"` is greater than a threshold (say, 5)?

```
python
CopyEdit
```

```
filtered = unique_combinations[unique_combinations["Count"] > 5]
print(filtered)
```

Let me know if you want help with any of these!

Otherwise, **brilliant job**—you're thinking and coding like a Digital Humanist already.

AI used in tri-grams script:

I have a dataset with yearly counts of specific 3-grams, and I'm creating a grouped bar chart showing the relative frequency of these phrases by year. Currently, the chart shows data starting from 2017, but for my analysis, I only want to include data from 2021 onwards to better align with my argument. How can I filter or subset my data to exclude earlier years before plotting, while still keeping the normalization calculations accurate? I want to understand the best approach conceptually, without just copying code.

That's a great mindset, Iqra — learning how to control your code instead of just copying it. Let's walk through how you can adjust the data so that your plot starts from **2021 onward**, step by step.

🎯 First, Understand Where the Problem Lies

You mentioned that the chart currently starts from 2017. This means your dataset contains records starting from 2017, and **you're currently plotting everything**. So the plot includes years that may not support the argument you're making (i.e., events that intensified post-2021).

So ask yourself:

Where is the chart getting the years from?

Answer: It's from the **`filtered` DataFrame**, specifically the `year` column.

☑ Step-by-Step Thought Process

Let's think about **how** to remove earlier years from this `filtered` DataFrame **before plotting**.

🧠 *Step 1: When is the right time to filter by year?*

You don't want to remove data from the original `data` DataFrame (in case you use it elsewhere), but you **can filter the `filtered` DataFrame** right before you create the plot.

Ask yourself:

Should I filter before or after calculating relative frequencies?

Answer: Filter **after** relative frequencies are calculated. Why?
Because your calculation uses `groupby('year')`, and if you remove years **before**, your relative frequencies will no longer be accurate (they'll be based only on the selected years, not all).

🧠 *Step 2: How do I restrict data to 2021 and after?*

You'll need to apply a condition:

Keep rows in `filtered` where the year is **greater than or equal to 2021**.

To think this through:

- What kind of object is the `year` column? Probably an **integer** (e.g., 2017, 2021).
- So you can apply a condition like: `filtered[filtered['year'] >= 2021]`

Just make sure this filtering happens **after** the relative frequency column is created, but **before** you send `filtered` into `px.bar()`.

🧠 *Step 3: What if your year column is a string?*

It *might* be a string depending on your CSV. So you should always check:

```
python
CopyEdit
print(filtered['year'].dtype)
```

If it's `object` or `string`, convert it using:

```
python
CopyEdit
filtered['year'] = filtered['year'].astype(int)
```

You only need to do this **once** before filtering.

💡 Recap

So in your own words:

1. I create a new DataFrame `filtered` that includes just my chosen trigrams.
2. I compute relative frequencies across all years.
3. Then, to plot only data from 2021 onward, I apply a condition to keep only rows where `year >= 2021`.
4. I do this before passing `filtered` to `px.bar()`.

When you're ready, try implementing this logic and let me know what happens — and if you hit a bump, I'll help you troubleshoot.

Hi ChatGPT, I'm working with a topic modeling dataset in Python using Pandas and Plotly Express. I have a DataFrame with columns for topic numbers (Topic) and the top 4 topic words (topic_1, topic_2, topic_3, topic_4). I want to create a bar chart that shows: The y-axis should display only the topic numbers (e.g., 13, 24, 44, 73). Each bar should display the combined topic label (the four topic words joined by commas) as text inside or on top of the bars. Guide me through it and do not give the codes directly, make me learn the

process. I already created a Topic_Label column by joining the four topic word columns. But when I try to plot, the chart either shows topic numbers or topic labels on the y-axis, but not both as I want. How can I plot a bar chart in Plotly Express where the y-axis shows only the topic numbers and the bars display the full topic labels as text? Please provide a minimal, clear example of how to do this with Pandas and Plotly Express. Thanks!
2/2

ChatGPT said:

Hey! Great you've already created a `Topic_Label` by joining those four words — that's the right first step. Let's break down the problem and how you can get the desired bar chart with Plotly Express:

What you want:

- **Y-axis:** Only the topic numbers (e.g., 13, 24, 44, 73).
- **Bars:** Length depends on some value (you can use counts or just a fixed value if it's just labels).
- **Bar text:** Show the combined topic words (your `Topic_Label`) inside or on top of each bar.
- **No replacing y-axis labels** — keep topic numbers on y-axis ticks.

Why your current approach shows either topic numbers or topic labels on y-axis:

Plotly Express uses the column mapped to the y-axis as the axis labels. If you set `y=Topic_Label`, you get topic labels on the axis; if you set `y=Topic` (topic numbers), then the axis shows numbers, but the text (if you use that for the bars) is a different argument.

You want to:

- Use `Topic` for y-axis (so those numbers show as tick labels)
- Use `Topic_Label` for the bar text (which is a separate argument)

Step-by-step thought process:

1. **What is your x-axis?**
   You need some numeric measure for the bar length (x-axis). If you don't have one, you can just use a constant number (like 1) for all bars, just to draw bars so you can show the text.
2. **Set y-axis to topic numbers:**
   So `y='Topic'` (your numeric or categorical topic number).
3. **Set x-axis to numeric values:**
   This can be your metric or just a dummy value like 1 if you just want bars of equal length.
4. **Use `text` argument for bar labels:**
   Pass your combined label column (`Topic_Label`) as the `text` argument.
5. **Control text position:**
   You can adjust `textposition` to `'inside'`, `'outside'`, or `'auto'` to get the text where you want it on the bars.

Minimal conceptual example:

Suppose your DataFrame looks like this:

Topic	topic_1	topic_2	topic_3	topic_4	Topic_Label
13	data	analysis	python	model	data, analysis, python, model
24	health	care	system	policy	health, care, system, policy
44	climate	change	carbon	emission	climate, change, carbon, emission
73	machine	learning	AI	algorithm	machine, learning, AI, algorithm

You want a bar chart with `Topic` on y-axis, bars of length 1, and the `Topic_Label` shown as text inside or on top of bars.

How would you think about building the chart?

- Set x=1 for all rows (or your real measure)

- Set `y='Topic'`
- Set `text='Topic_Label'`
- Choose `orientation='h'` to have horizontal bars (Topic on y-axis, bars go along x)
- Set `textposition='inside'` (for example)

Why horizontal bars?

Horizontal bars are easier for displaying long text labels on the bars alongside categorical y-axis.

Your turn:

- Do you have a numeric value for bar length, or do you want equal-length bars just to display labels?
- Do you want bars horizontal (topics on y) or vertical (topics on x)?
- How do you want the text positioned relative to the bars?

Once you confirm these, I can guide you on how to set parameters in Plotly Express, step by step, without giving direct code just yet. Sound good?

☑ Mini-Project 3 Report: Trigram and Topic Modeling Analysis of the Al-Jazeera Gaza Corpus

📌 Objective of This Analysis

This project aimed to analyze the **Al-Jazeera Gaza conflict corpus** through both **trigram frequency trends** and **topic modeling**, allowing us to examine not only *what* kinds of phrases were repeated over time but also the *deeper thematic structures* beneath them.

Our goals were:

1. To explore how specific conflict-related trigrams changed across time.
2. To see how these phrases aligned with larger thematic clusters generated through topic modeling.
3. To reflect critically on the process of filtering, analyzing, and interpreting the data.

🧠 Our Journey: Thought Process, Dataset Filtering, and Methodological Choices

From the beginning, I was conscious of the need to **document the thought process** and **filter choices** in a transparent, research-oriented way.

Dataset Selection

- I chose `3-gram-year.csv` for trigram analysis because it provided **temporal (year-wise)** frequency data of trigrams, which aligned with our aim to observe **yearly trends**.
- For deeper themes, I selected `topic-model.csv`, which offered **topic numbers and their associated words**, enabling latent theme analysis.

What We Were Trying to Discover

- I wanted to see if phrases like “war on gaza” or “told al jazeera” reflected **shifting media tone**, and if these trigrams correlated with **underlying thematic clusters** in topic modeling.
- I was also curious whether **certain actors (e.g., “israeli army”) or outcomes (“have been killed”)** were consistent in framing.

Filtering Criteria

- From hundreds of trigrams, I filtered only those that were both **topical** and **recurrent**.
- In topic modeling, I filtered topic numbers that contained **overlapping or semantically similar words** with selected trigrams (e.g., “gaza”, “rocket”, “army”, “killed”).

Obstacles Faced

- **Messy Data**: Some CSVs were not cleanly formatted. Inconsistent column naming and missing values made merging or filtering difficult.
- **Topic Inconsistency**: Topic numbers weren't labeled by theme, so I had to **manually interpret** which ones related to my trigrams—requiring subjective judgment.
- **Visualizing Cleanly**: Plotting normalized bar graphs and ensuring color distinctions across years and topics took significant trial and error.

What If the Data Had Been Cleaner?

- A more refined topic model CSV—with **pre-labeled themes or cleaner tokenization**—would've helped greatly.
- If trigram files had **frequency per million words** or cleaner tags (e.g., source, location), I could have made more nuanced claims.

📊 Visualization Insights

A. Trigram Frequency Chart (2021–2024)

Key observations:

- "**Told al jazeera**" appeared consistently and most frequently, showing **reporting voice centrality**.
- "**War on gaza**" jumped sharply in 2024, potentially reflecting an escalation or shift in framing.
- "**Have been killed**" showed moderate but consistent presence, pointing to casualty-focused reporting.
- "**Hamas rocket attacks**" was surprisingly rare, suggesting either editorial caution or use of alternate phrasing.

B. Topic Modeling Chart Analysis

(See Screenshot 125)

- Topics 12, 19, 33, and 58 were chosen due to semantic overlap with selected trigrams.
- **Topic 12 (73 articles)** emerged as the most frequent, with likely themes of **conflict, attacks, and humanitarian crises**.
- Topic 58, while the least common, still captured a distinct dimension possibly tied to military events or diplomacy.

🔍 Analytical Reflections and Unexpected Discoveries

Was My Central Concern Answered?

- **Yes**, to a large extent. The graph clearly showed how specific phrases and themes evolved across time and coalesced around war and media framing.
- Topic modeling helped **validate** that these trigrams were not isolated — they were embedded within larger thematic fields.

Any Surprises?

- **Underrepresentation of "hamas rocket attacks"** was unexpected. I had assumed it would be more prevalent.
- The **dominance of "told al jazeera"** was also more striking than I anticipated—it shows how much direct attribution shapes Al-Jazeera's reporting.

Intellectual Insight Gained

- The project helped me realize that **distant reading isn't just about frequencies or topics**—it's about triangulating between different layers of discourse.
- The tension between **data messiness and interpretive clarity** taught me that **qualitative judgment still plays a major role** in digital humanities research.

☑ Conclusion: Portfolio Relevance

This project combined:

- **Quantitative fluency** in trigram and topic modeling.
- **Critical reflection** on the research process.
- **Clear visual storytelling** to support interpretive claims.

By layering trigrams and topic modeling—and documenting my journey—I was able to move from **surface language patterns** to **deep thematic analysis**, making this not just a technical exercise but also a reflective scholarly endeavor.