**AI USED FOR**

* **Checking if the code fulfills the criteria.**

**You said:**

**CHECK THE CODE FOR THE FOLLOWING ● Well-documented scripts (with comments throughout) to filter the data, analyse it and produce visualisations ● Understanding of the appropriate use of a visualisation to assert a point ● Appropriate formatting of a visualisation to assert the point (lack of clutter, clear labels, and clear annotation where needed) # Importing Required Libraries import pandas as pd import plotly.express as px import os # Loading the Dataset # Setting base directory path base\_dir = os.path.join("..") # Move one level up if scripts and data are siblings # Define the full path to the topic modelling CSV topic\_model\_path = os.path.join(base\_dir, "data", "dataframes", "topic-model", "topic-model.csv") # Load the topic modelling CSV file using pandas df = pd.read\_csv(topic\_model\_path) # Clean and Prepare the Data # Filter out rows where Topic == -1 (unassigned articles) df = df[df['Topic'] != -1] # Create a new column with a human-readable topic label using the top 4 keywords # This will help us understand what each topic is about df['Topic\_Keywords'] = df[['topic\_1', 'topic\_2', 'topic\_3', 'topic\_4']].agg(', '.join, axis=1) # Count Articles per Topic # Group by Topic and count the number of articles per topic topic\_counts = df.groupby(['Topic', 'Topic\_Keywords']).size().reset\_index(name='Article\_Count') # Sort by the most frequent topics topic\_counts = topic\_counts.sort\_values(by='Article\_Count', ascending=False) # Calculate percentage to show how much media coverage each topic occupies total\_articles = topic\_counts['Article\_Count'].sum() topic\_counts['Percentage'] = (topic\_counts['Article\_Count'] / total\_articles \* 100).round(2) # Visualise: Top N Topics by Frequency # For better readability, we WILL focus on the top 10 topics only top\_n = 10 top\_topics = topic\_counts.head(top\_n).copy() # Add labels for better argument-making top\_topics['Label'] = top\_topics['Topic\_Keywords'] + '\n(' + top\_topics['Percentage'].astype(str) + '% of coverage)' # Create a horizontal bar chart using Plotly Express fig = px.bar( top\_topics, x='Article\_Count', y='Label', orientation='h', title=f"📰 Top {top\_n} Topics in al-Jazeera Gaza Corpus\n(Showing media focus and thematic saturation)", labels={'Article\_Count': 'Number of Articles', 'Label': 'Topic (Top 4 Keywords)'}, height=700 ) # Improve the layout for readability fig.update\_layout(yaxis=dict(autorange="reversed")) # Highest on top fig.update\_traces(marker\_color='crimson') # Optional: Red color to match urgency/conflict # Show the plot fig.show() # Save Output # Define output path for topic summary CSV output\_path = os.path.join(base\_dir, "data", "dataframes", "topic-model", "topic\_summary.csv") # Save the sorted topic counts to a CSV file for reference topic\_counts.to\_csv(output\_path, index=False)**

**ChatGPT said:**

**Your code is overall well-structured and meets the requirements. Here's an evaluation based on the specified criteria:**

**✅ 1. Well-Documented Scripts**

**Evaluation: Excellent**

**Strengths:**

* **✅ Inline comments clearly explain each major step, such as data loading, filtering, grouping, and visualization.**
* **✅ Descriptive variable names (topic\_counts, top\_topics, Label) enhance readability.**
* **✅ Use of comments like # For better readability, we will focus on the top 10 topics only reflects intentionality in design.**

**Suggestions (Minor):**

* **You could briefly explain what topic\_1 to topic\_4 represent for context (e.g., "keywords from topic modeling").**

**✅ 2. Understanding of the Appropriate Use of a Visualisation**

**Evaluation: Strong**

**Strengths:**

* **✅ You chose a horizontal bar chart, which is ideal for comparing categories (topics) by count.**
* **✅ The inclusion of percentage labels in the chart aids interpretation by contextualizing raw counts.**
* **✅ The visualisation supports the point of media focus and thematic saturation, aligning with the research aim.**

**Suggestions:**

* **Consider emphasizing whether some topics were disproportionately covered in your conclusion/annotation if this supports a critical media analysis.**

**✅ 3. Appropriate Formatting of Visualisation**

**Evaluation: Strong**

**Strengths:**

* **✅ Clean layout: no clutter or 3D elements.**
* **✅ Clear y-axis labels combining keywords and percentages.**
* **✅ Color (crimson) supports the emotional tone of conflict coverage.**
* **✅ Title includes both source ("al-Jazeera Gaza Corpus") and chart purpose.**

**Suggestions (Optional Enhancements):**

* **Add annotations or a subtitle to highlight the most covered topic (e.g., “Topic X received the most attention: Y% of all articles”).**
* **Consider increasing font size for better legibility, especially for presentations or reports.**

**✅ Summary**

| **Criterion** | **Rating** | **Notes** |
| --- | --- | --- |
| **Documentation** | **✅ Excellent** | **Clear, logical, and thorough comments** |
| **Appropriate Visualization** | **✅ Strong** | **Chart type and labels align with purpose** |
| **Formatting & Clarity** | **✅ Strong** | **Clean design, meaningful color, well-labeled** |

**🚀 Optional Enhancements**

**If you want to make the script even stronger:**

* **📌 Add a subtitle to the plot using fig.update\_layout(title=dict(text=..., font=dict(...))).**
* **🧪 Consider saving the Plotly chart to HTML (with fig.write\_html) for reproducibility.**
* **🗂️ Add logging (e.g., logging.info) for larger projects or when integrating multiple scripts.**

**Let me know if you want help adding any of those enhancements!**

* **Topic Modelling and Keyword Aggregation - Identifying Core Themes in the Corpus**  
    
  I need to Aggregate the most frequent keyword in each topic for four keyword columns, what ways can i use?

**ChatGPT said:**

To aggregate the **most frequent keyword for each topic** across **four keyword columns** (e.g., topic\_1, topic\_2, topic\_3, topic\_4), you can use several approaches depending on what exactly you mean by “most frequent”:

**✅ Goal: For each topic, find the most commonly occurring keyword *overall* across the four keyword columns.**

Let’s assume your DataFrame is called df and has a Topic column and four keyword columns: topic\_1, topic\_2, topic\_3, and topic\_4.

**🔹 Method 1: Melt + GroupBy + Mode (most common keyword overall)**

python

CopyEdit

# Reshape to long format

keyword\_long = df.melt(id\_vars='Topic', value\_vars=['topic\_1', 'topic\_2', 'topic\_3', 'topic\_4'],

var\_name='Position', value\_name='Keyword')

# Group by topic and get the most frequent keyword (mode)

most\_frequent = keyword\_long.groupby('Topic')['Keyword'].agg(lambda x: x.mode().iloc[0]).reset\_index()

most\_frequent.columns = ['Topic', 'Most\_Frequent\_Keyword']

**🔹 Method 2: Count occurrences using value\_counts()**

python

CopyEdit

# Same reshaping

keyword\_long = df.melt(id\_vars='Topic', value\_vars=['topic\_1', 'topic\_2', 'topic\_3', 'topic\_4'],

var\_name='Position', value\_name='Keyword')

# Count occurrences per keyword per topic

keyword\_counts = keyword\_long.groupby(['Topic', 'Keyword']).size().reset\_index(name='Count')

# Select the most frequent keyword per topic

most\_frequent = keyword\_counts.sort\_values(['Topic', 'Count'], ascending=[True, False]) \

.drop\_duplicates('Topic')

**🔹 Method 3: Use mode on each keyword column, then select most frequent overall**

This is less precise for your goal, but for completeness:

python

CopyEdit

# Aggregate modes individually

modes\_df = df.groupby('Topic')[['topic\_1', 'topic\_2', 'topic\_3', 'topic\_4']].agg(lambda x: x.mode()[0])

# Optionally combine all 4 into a frequency count

**🔹 Bonus: Get all keywords ranked by frequency per topic**

If you want a frequency *ranking* instead of just the top one:

python

CopyEdit

# Melt as before

keyword\_long = df.melt(id\_vars='Topic', value\_vars=['topic\_1', 'topic\_2', 'topic\_3', 'topic\_4'],

var\_name='Position', value\_name='Keyword')

# Count and sort

keyword\_freqs = keyword\_long.groupby(['Topic', 'Keyword']).size().reset\_index(name='Count') \

.sort\_values(['Topic', 'Count'], ascending=[True, False])

Let me know which one fits best and I can turn it into a reusable function or DataFrame.

NO. PRECISELY ANSWER THIS QUESTION How can I aggregate the most common keyword in each topic for the columns topic\_1 to topic\_4 using pandas?

**ChatGPT said:**

Here’s how you can aggregate the most common keyword in each topic for the columns topic\_1 to topic\_4 using pandas:

python

CopyEdit

top\_words = (

df\_clean.groupby("Topic")[["topic\_1", "topic\_2", "topic\_3", "topic\_4"]]

.agg(lambda x: pd.Series.mode(x)[0]) # Select the most common value in each column

.reset\_index()

)

what does lambda does?

**ChatGPT said:**

In this code, the lambda function is an **anonymous (inline) function** that takes an input x and returns the most common value (mode) within it.

Specifically:

* x is a pandas Series representing all the values of one keyword column (e.g., topic\_1) within a single topic group.
* pd.Series.mode(x) returns the mode(s) — the most frequently occurring value(s) in that Series.
* [0] selects the first mode if there are multiple.
* So, the lambda returns the single most common keyword for that column and topic.

In short, the lambda here helps to **find the top (most frequent) keyword in each column for each topic** during the group aggregation.

how is this code combining key words for each topic? and on what bases is it doing it?

**ChatGPT said:**

Great question — understanding **how and why** the keywords are being combined helps clarify both the logic and the purpose behind that part of the code.

**🔍 How the Code Combines Keywords**

This line does the combination:

python

CopyEdit

df['Topic\_Keywords'] = df[['topic\_1', 'topic\_2', 'topic\_3', 'topic\_4']].agg(', '.join, axis=1)

**What it does:**

* It selects four columns: topic\_1, topic\_2, topic\_3, and topic\_4.
* These columns contain the **top 4 keywords** for each topic, typically generated during the **topic modeling process** (like with LDA or NMF).
* agg(', '.join, axis=1) tells pandas to:
  + Go across each row (axis=1 means row-wise),
  + Combine the four keyword strings using a comma and space as the separator.

**Example:**

If a row has:

python

CopyEdit

topic\_1: "Israel"

topic\_2: "Gaza"

topic\_3: "Hamas"

topic\_4: "war"

Then the new column Topic\_Keywords will have:

arduino

CopyEdit

"Israel, Gaza, Hamas, war"

**📌 On What Basis Are These Keywords Chosen?**

These keywords come from the **topic modeling algorithm** used *before* this code — usually tools like:

* **Latent Dirichlet Allocation (LDA)**
* **Non-negative Matrix Factorization (NMF)**

During topic modeling, each topic is represented by a distribution of words. The top N (here, 4) words with the highest weight or probability in that distribution are extracted as **topic descriptors**.

These keywords:

* Are not selected manually.
* Represent the **most important or frequent terms** in that topic across the corpus.
* Help humans understand what the topic is "about."

**✅ Why Combine Them?**

Combining them into one readable string:

* Helps label and interpret each topic easily in charts or tables.
* Makes it possible to display a **descriptive topic name** without needing complex logic or long descriptions.

If you'd like, I can show you the earlier step where the topic model typically generates those topic\_1 through topic\_4 keywords — or help you improve how those are labeled (e.g., using different numbers of keywords or custom naming).

* **For knowing if I could work with all the articles (without creating a dataframe). This was just out of curiosity.**

what if we added articles paths? will it help it in any way?

**ChatGPT said:**

Great question — adding article paths **can** help, but it depends on **what you want to do**.

**🔹 If You Only Want to Visualize Topic Frequencies (Like This Code Does):**

**Adding article paths won't make a difference.**  
This code is only interested in **how many articles** belong to each topic, and what **keywords** describe those topics — not the article content itself.

So adding a column like article\_path or file\_path won’t change what this script produces unless you specifically want to:

* **Display example articles for each topic** (e.g., "here are 2 articles from Topic 3").
* **Link articles in a dashboard or web app**.
* **Let users read articles related to a given topic**.

**🔹 When Adding Article Paths *Would* Help:**

You can **enhance** this script or future analysis by including article paths if you plan to:

1. **Manually inspect articles from each topic**  
   → Useful for checking if topic labels make sense.
2. **Sample or quote articles in a report or presentation**  
   → You can quickly go to the source file using the path.
3. **Do further NLP or text analysis**  
   → E.g., summarizing each topic by finding representative paragraphs from the original articles.
4. **Build an interactive dashboard**  
   → Show article previews or full texts on click.