Analyzing Jokerace Submissions: Vision, Mission, and Strategic Priorities

July 29, 2025

Author: Omniacs.DAO

Case Study: From Raw Text to Actionable Insights with Python and OpenAI

How can you make sense of thousands of free-text submissions from the Arbitrum community? This is a common challenge for decentralized autonomous organizations (DAOs), companies, and projects that rely on community feedback to guide their strategy. Raw text is messy, unstructured, and time-consuming to analyze manually.

In this case study, we'll walk through a complete, real-world workflow for analyzing community proposal submissions from the Arbitrum DAO's Jokerace contests. We will use a powerful combination of classic NLP techniques and modern AI to extract meaningful themes and generate a high-level summary.

You will learn how to:

1. **Preprocess and Clean** raw text data for analysis.
2. Use **N-gram Analysis** to find common multi-word phrases.
3. Apply **Topic Modeling (LDA)** to discover latent themes in the submissions.
4. Leverage the **OpenAI API** to synthesize your findings into a concise, human-readable report.

1 Project Overview and Setup

Objective

Plurality Labs, founded by @DisruptionJoe and supported by his teammates @prose11 | GovAlpha and @radioflyerMQ were given the task to setup experimental governance mechanisms that would allow for formalized "sense-making" that will help the DAO come up with a strong mission and vision to guide their decisions as they then follow up with initiating the capital allocation process where grants will be given out to support the ARB ecosystem.

One of those experiments was an online incentivized survey hosted by "JokerRace" as part of the ThanksARB initiative of Arbitrum's "#GOVMonth". The Jokeraces consisted of 4 separate on-chain surveys where users were prompted to provide feedback on:

- ArbitrumDAO Short-term Strategic Priorities (Reduce Friction) - `0xbf47BDA4b172daf321148197700cBED04dbe0D58`
- ArbitrumDAO Long-term Strategic Priorities (Growth and Innovation) - `0x5D4e25fA847430Bf1974637f5bA8CB09D0B94ec7`
- ArbitrumDAO Northstar Strategic Framework: Vision - `0x0d4C05e4BaE5eE625aADC35479Cc0b140DDF95D4`
- ArbitrumDAO Northstar Strategic Framework: Mission - `0x5a207fA8e1136303Fd5232E200Ca30042c45c3B6`

The goal of this case-study is to analyze submissions across four Jokerace contests—Vision, Mission, Short-term priorities, and Long-term priorities—to:

1. Identify key themes, concerns, and strategic ideas proposed by the community.
2. Develop a reusable methodology for future text analysis.
3. Create a summarized report of the findings.

Python Environment Setup

First, let's set up our Python environment by loading the necessary libraries. These packages cover everything from data manipulation (`pandas`) and text mining (`nltk`, `scikit-learn`) to API communication (`requests`).

To install the required packages, run:

```
pip install pandas scikit-learn nltk beautifulsoup4 requests
```

[1]:
```
# --- Load Libraries ---
# For data manipulation
import pandas as pd
import numpy as np
import re
import os

# For text mining and NLP
from bs4 import BeautifulSoup
import nltk
from nltk.stem import PorterStemmer
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation

# For interacting with APIs
import requests
import json

# For display purposes in Jupyter
from IPython.display import display, HTML

# --- Initial Setup ---
# Download necessary NLTK data (if not already downloaded)
```

```
try:
    stopwords.words('english')
except LookupError:
    nltk.download('stopwords')

# Initialize stemmer
stemmer = PorterStemmer()
```

Loading the Data

We'll use one main dataset: `propfinal`: Contains the core proposal submissions, including the text content, author, and timestamp from JokerAce.

(Note: The original `.RDS` file has been converted to a more Python-friendly `.csv` format for this notebook.)

[2]:
```
# Load Data
# The original R script used readRDS. We'll use pandas to read a CSV or Pickle␣
 ↪file.
# Ensure the data file 'DataProposalsJokerAce.csv' is in a '../data/' directory␣
 ↪relative to this notebook.
try:
    propfinal = pd.read_csv('../data/DataProposalsJokerAce.csv')
except FileNotFoundError:
    print("Error: Data file not found. Please ensure 'DataProposalsJokerAce.csv'␣
 ↪is in the '../data/' directory.")
    # Create a dummy dataframe for demonstration purposes if file doesn't exist
    propfinal = pd.DataFrame({
        'Content': ['<p>Example submission 1 about fixing bugs.</p>', 'Another␣
 ↪entry about community growth.', 'Improve the user experience by reducing fees.
 ↪'],
        'Contract': ['0xbf47bda4b172daf321148197700cbed04dbe0d58',␣
 ↪'0x5d4e25fa847430bf1974637f5ba8cb09d0b94ec7',␣
 ↪'0xbf47bda4b172daf321148197700cbed04dbe0d58'],
        'ContentParsed': ['Example submission 1 about fixing bugs.', 'Another␣
 ↪entry about community growth.', 'Improve the user experience by reducing fees.
 ↪']
    })

display(propfinal.head())
```

```
             BlockTime                                       Contract  \
0  2023-09-06T09:12:57Z  0xbf47bda4b172daf321148197700cbed04dbe0d58
1  2023-09-06T09:13:36Z  0x5d4e25fa847430bf1974637f5ba8cb09d0b94ec7
2  2023-09-06T09:14:02Z  0x0d4c05e4bae5ee625aadc35479cc0b140ddf95d4
3  2023-09-06T09:14:34Z  0x5a207fa8e1136303fd5232e200ca30042c45c3b6
4  2023-09-09T11:34:17Z  0x5a207fa8e1136303fd5232e200ca30042c45c3b6

                                    Address  \
0  0x6899dd4c3ceab23e7022ff5ead9b9e9c78333105
```

```
1  0x6899dd4c3ceab23e7022ff5ead9b9e9c78333105
2  0x6899dd4c3ceab23e7022ff5ead9b9e9c78333105
3  0x6899dd4c3ceab23e7022ff5ead9b9e9c78333105
4  0x88496a9b5d7543148d9f8aecc9f69cb8437a4bae

                                                  Slug  \
0  587785712853410558465483868824947201479248167 2...
1  327148190878820206313920007630368003569625152 1...
2  794349898572753675235530281904430798570851323 2...
3  222349055254978359128534916227957123274176027 7...
4  585503403010717422117549964403530159039833814 0...

                                                   URL  IsImage  \
0  https://jokerace.xyz/_next/data/k6jlVObdmnNyS2...    False
1  https://jokerace.xyz/_next/data/k6jlVObdmnNyS2...    False
2  https://jokerace.xyz/_next/data/k6jlVObdmnNyS2...    False
3  https://jokerace.xyz/_next/data/k6jlVObdmnNyS2...    False
4  https://jokerace.xyz/_next/data/k6jlVObdmnNyS2...    False

                                               Content  \
0  <p>Arbitrum DAO's short term is to enhance acc...
1  <p>Arbitrum DAO's long term is to establish a ...
2  <p>Arbitrum DAO's vision is to revolutionize b...
3  <p>Arbitrum DAO's mission is to drive innovati...
4                                 <p>cxvxcvfds</p>

                                       ContentParsed
0  Arbitrum DAO's short term is to enhance access...
1  Arbitrum DAO's long term is to establish a rob...
2  Arbitrum DAO's vision is to revolutionize bloc...
3  Arbitrum DAO's mission is to drive innovation,...
4                                          cxvxcvfds
```

2 Data Cleaning and Preprocessing

Text data from the wild is messy. It's filled with HTML tags, punctuation, and inconsistent capitalization. To prepare it for analysis, we need to standardize it through a process called preprocessing. Our pipeline will:

1. Remove HTML tags.
2. Convert all text to lowercase.
3. Remove punctuation.
4. Stem words to their root form (e.g., "running", "ran", and "runs" all become "run"). This helps group related words.
5. Trim excess whitespace.

[3]:
```
# --- Clean and Preprocess Text ---

def clean_text(text):
    # 1. Remove HTML tags
    text = BeautifulSoup(text, "html.parser").get_text()
    # 2. Convert to lowercase
    text = text.lower()
    # 3. Remove punctuation
    text = re.sub(r'[^\w\s]', '', text)
    # 4. Stem words
    text = ' '.join([stemmer.stem(word) for word in text.split()])
    # 5. Trim whitespace
    return text.strip()

# Apply the cleaning function to the 'Content' column
propfinal_clean = propfinal['Content'].apply(clean_text)

# Display a before-and-after comparison
df_preview = pd.DataFrame({
    'Raw': propfinal['Content'].head(),
    'Cleaned': propfinal_clean.head()
})

display(df_preview)
```

```
                                                 Raw  \
0  <p>Arbitrum DAO's short term is to enhance acc...
1  <p>Arbitrum DAO's long term is to establish a ...
2  <p>Arbitrum DAO's vision is to revolutionize b...
3  <p>Arbitrum DAO's mission is to drive innovati...
4                                <p>cxvxcvfds</p>

                                              Cleaned
0  arbitrum dao short term is to enhanc access us...
1  arbitrum dao long term is to establish a robus...
2  arbitrum dao vision is to revolution blockchai...
3  arbitrum dao mission is to drive innov foster ...
4                                            cxvxcvfd
```

This clean, stemmed text is now ready for more advanced analysis.

3 Exploratory Analysis: N-grams

What are the most common phrases in the submissions? While looking at single words (unigrams) is useful, n-grams (sequences of n words) give us more context. For example, the trigram "reduce transaction fees" is far more insightful than the individual words "reduce," "transaction," and "fees."

Let's find the most common trigrams (3-word phrases), excluding common "stop words" like "the," "a," and "is". We'll also add custom stop words specific to our dataset, like "arbitrum" and "dao," to filter out noise.

[4]:
```
# --- N-gram Analysis ---

# Define stopwords
custom_stopwords = ['arbitrumdao', 'arbitrum', 'project', 'arb', 'dao']
stop_words = list(stopwords.words('english')) + custom_stopwords

# Use CountVectorizer to get trigram counts
vec = CountVectorizer(ngram_range=(3, 3), stop_words=stop_words).
 ↪fit(propfinal_clean)
bag_of_words = vec.transform(propfinal_clean)
sum_words = bag_of_words.sum(axis=0)
words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
trigram_table = sorted(words_freq, key = lambda x: x[1], reverse=True)

# Format as a DataFrame for display
trigram_df = pd.DataFrame(trigram_table, columns=['Trigram', 'Count'])

print("Top 15 Trigrams:")
display(trigram_df.head(15))
```

```
Top 15 Trigrams:
```

	Trigram	Count
0	make wise decis	212
1	believ make wise	211
2	agre becaus believ	207
3	becaus believ make	206
4	read agre posit	136
5	agre posit thing	131
6	layer scale solut	122
7	creat anti sybil	100
8	shortterm strateg prioriti	96
9	love hope soon	93
10	longterm strateg prioriti	86
11	thi fantast great	86
12	agre make develop	80
13	think make develop	79
14	glad part thi	79

This output immediately gives us a feel for recurring suggestions and ideas within the community.

4 Uncovering Themes with Topic Modeling (LDA)

While n-grams show us popular phrases, topic modeling helps us discover the underlying, latent themes across all submissions. We'll use Latent Dirichlet Allocation (LDA), a popular unsuper-

vised algorithm that works by:

1. Assuming each document is a mix of topics.
2. Assuming each topic is a mix of words.
3. Figuring out the "topics" (which are just clusters of words) that best explain the collection of documents.

The LDA Workflow

We'll create a repeatable workflow to process each contest's submissions.

Step 1: Define Helper Functions

We'll create functions to vectorize the text into a Document-Term Matrix (DTM) and run the LDA model. A DTM is a simple matrix where rows represent documents, columns represent terms (words), and the cells contain the word counts.

[5]:
```
# --- LDA Helper Functions ---

def create_dtm(texts, stop_words_list, min_freq=5):
    """Creates a Document-Term Matrix from a list of texts."""
    vectorizer = CountVectorizer(
        stop_words=stop_words_list,
        min_df=min_freq, # equivalent to global bounds in tm
        preprocessor=lambda x: re.sub(r'\d+', '', x.lower()), # remove numbers␣
 ↪and lowercase
        tokenizer=lambda x: [stemmer.stem(w) for w in re.split(r'\W+', x) if w]␣
 ↪# stem and tokenize
    )
    dtm = vectorizer.fit_transform(texts)
    # Filter out documents that become empty after preprocessing
    non_empty_docs_mask = np.asarray(dtm.sum(axis=1)).flatten() > 0
    return dtm[non_empty_docs_mask, :], vectorizer, non_empty_docs_mask

def run_lda(dtm, K, iterations=50):
    """Runs the LDA algorithm."""
    lda_model = LatentDirichletAllocation(
        n_components=K,
        random_state=1234,
        max_iter=iterations,
        learning_method='online' # similar to 'Gibbs'
    )
    lda_model.fit(dtm)
    return lda_model

def get_top_terms(model, feature_names, n=20):
    """Extracts top terms from the LDA model."""
    top_terms = {}
    for topic_idx, topic in enumerate(model.components_):
```

```
        top_terms[f"Topic_{topic_idx + 1}"] = [feature_names[i] for i in topic.
↪argsort()[:-n - 1:-1]]
    return pd.DataFrame(top_terms)
```

Step 2: Set Up and Run the Analysis Loop

Now, we define the contests we want to analyze and the number of topics (K) we want to find for each. Choosing K is a mix of art and science; it often requires experimentation to find a number that produces coherent, distinct topics.

The loop will perform the following for each contest:

1. Subset the proposals for that contest.
2. Perform final cleaning and filtering.
3. Create the DTM using our helper function.
4. Run the LDA model.
5. Assign the most likely topic to each proposal.
6. Extract and save the top words for each discovered topic.

[6]:
```
# --- Define Contests and K (Number of Topics) ---
contest_data = [
  {'Address': "0xbf47bda4b172daf321148197700cbed04dbe0d58", 'Name': "Reduce␣
↪Friction", 'Slug': "RF4", 'Topics': 4},
  {'Address': "0x5d4e25fa847430bf1974637f5ba8cb09d0b94ec7", 'Name': "Growth and␣
↪Innovation", 'Slug': "GI5", 'Topics': 5},
  {'Address': "0x0d4c05e4bae5ee625aadc35479cc0b140ddf95d4", 'Name': "Vision",␣
↪'Slug': "V7", 'Topics': 7},
  {'Address': "0x0d4c05e4bae5ee625aadc35479cc0b140ddf95d4", 'Name': "Vision",␣
↪'Slug': "V10", 'Topics': 10},
  {'Address': "0x5a207fa8e1136303fd5232e200ca30042c45c3b6", 'Name': "Mission",␣
↪'Slug': "M11", 'Topics': 11},
  {'Address': "0x5a207fa8e1136303fd5232e200ca30042c45c3b6", 'Name': "Mission",␣
↪'Slug': "M15", 'Topics': 15}
]
contestdf = pd.DataFrame(contest_data)

# Create a directory to store outputs
output_dir = "../project1/temp/txts"
for slug in contestdf['Slug']:
    os.makedirs(os.path.join(output_dir, slug), exist_ok=True)

# --- Main LDA Analysis Loop ---
all_results = []
propout = propfinal.copy()

for idx, contest in contestdf.iterrows():
    print(f"Processing: {contest['Name']} ({contest['Slug']})")
    # 1. Subset data for this contest
```

```
    subset_df = propfinal[propfinal['Contract'] == contest['Address']].copy()

    # 2. Clean and filter text
    # Fix encoding, filter by length, and remove duplicates
    subset_df['ContentParsed'] = subset_df['ContentParsed'].str.encode('ascii',␣
↪'ignore').str.decode('ascii')
    subset_df = subset_df[subset_df['ContentParsed'].str.len() > 40]
    subset_df = subset_df.drop_duplicates(subset=['ContentParsed'])

    textdata = subset_df['ContentParsed']

    if len(textdata) < contest['Topics']:
        print(f"  Skipping {contest['Slug']}: Not enough documents.")
        continue

    # 3. Create DTM
    dtm, vectorizer, valid_mask = create_dtm(textdata, stop_words, min_freq=5)
    docdf = subset_df.loc[valid_mask].copy()

    if dtm.shape[0] < contest['Topics']:
        print(f"  Skipping {contest['Slug']}: Not enough documents after pruning.
↪")
        continue

    # 4. Topic modeling
    K = contest['Topics']
    lda_model = run_lda(dtm, K)
    topic_results = lda_model.transform(dtm)
    docdf['Topic'] = topic_results.argmax(axis=1) + 1 # 1-based topic index

    # 5. Annotate main dataframe with topic assignment
    propout.loc[docdf.index, contest['Slug']] = docdf['Topic']

    # 6. Save per-topic submissions as text files
    for topic_num in docdf['Topic'].unique():
        tempd = docdf[docdf['Topic'] == topic_num]
        out_path = os.path.join(output_dir, contest['Slug'], f"Topic-{topic_num}.
↪txt")
        with open(out_path, 'w', encoding='utf-8') as f:
            for i, row in tempd.iterrows():
                f.write(f"Submission : {i}\n{row['ContentParsed']}\n\n\n")

    # 7. Save top terms
    top_terms_df = get_top_terms(lda_model, vectorizer.get_feature_names(), n=50)
    top_terms_df.columns = [f"{contest['Slug']}_{col}" for col in top_terms_df.
↪columns]
    all_results.append(top_terms_df)
```

```
# Combine all topic term results into a single dataframe
if all_results:
    results = pd.concat(all_results, axis=1)
    print("\nLDA Analysis Complete. Top terms extracted:")
    display(results.head())
else:
    print("LDA Analysis did not produce any results.")
```

```
Processing: Reduce Friction (RF4)

C:\Users\Business\anaconda3\lib\site-
packages\sklearn\feature_extraction\text.py:388: UserWarning: Your stop_words
may be inconsistent with your preprocessing. Tokenizing the stop words generated
tokens ['abov', 'ani', 'becaus', 'befor', 'doe', 'dure', 'ha', 'hi', 'onc',
'onli', 'ourselv', 'themselv', 'thi', 'veri', 'wa', 'whi', 'yourselv'] not in
stop_words.
  warnings.warn('Your stop_words may be inconsistent with '

Processing: Growth and Innovation (GI5)

C:\Users\Business\anaconda3\lib\site-
packages\sklearn\feature_extraction\text.py:388: UserWarning: Your stop_words
may be inconsistent with your preprocessing. Tokenizing the stop words generated
tokens ['abov', 'ani', 'becaus', 'befor', 'doe', 'dure', 'ha', 'hi', 'onc',
'onli', 'ourselv', 'themselv', 'thi', 'veri', 'wa', 'whi', 'yourselv'] not in
stop_words.
  warnings.warn('Your stop_words may be inconsistent with '

Processing: Vision (V7)

C:\Users\Business\anaconda3\lib\site-
packages\sklearn\feature_extraction\text.py:388: UserWarning: Your stop_words
may be inconsistent with your preprocessing. Tokenizing the stop words generated
tokens ['abov', 'ani', 'becaus', 'befor', 'doe', 'dure', 'ha', 'hi', 'onc',
'onli', 'ourselv', 'themselv', 'thi', 'veri', 'wa', 'whi', 'yourselv'] not in
stop_words.
  warnings.warn('Your stop_words may be inconsistent with '

Processing: Vision (V10)

C:\Users\Business\anaconda3\lib\site-
packages\sklearn\feature_extraction\text.py:388: UserWarning: Your stop_words
may be inconsistent with your preprocessing. Tokenizing the stop words generated
tokens ['abov', 'ani', 'becaus', 'befor', 'doe', 'dure', 'ha', 'hi', 'onc',
'onli', 'ourselv', 'themselv', 'thi', 'veri', 'wa', 'whi', 'yourselv'] not in
stop_words.
  warnings.warn('Your stop_words may be inconsistent with '

Processing: Mission (M11)

C:\Users\Business\anaconda3\lib\site-
```

```
packages\sklearn\feature_extraction\text.py:388: UserWarning: Your stop_words
may be inconsistent with your preprocessing. Tokenizing the stop words generated
tokens ['abov', 'ani', 'becaus', 'befor', 'doe', 'dure', 'ha', 'hi', 'onc',
'onli', 'ourselv', 'themselv', 'thi', 'veri', 'wa', 'whi', 'yourselv'] not in
stop_words.
  warnings.warn('Your stop_words may be inconsistent with '

Processing: Mission (M15)

C:\Users\Business\anaconda3\lib\site-
packages\sklearn\feature_extraction\text.py:388: UserWarning: Your stop_words
may be inconsistent with your preprocessing. Tokenizing the stop words generated
tokens ['abov', 'ani', 'becaus', 'befor', 'doe', 'dure', 'ha', 'hi', 'onc',
'onli', 'ourselv', 'themselv', 'thi', 'veri', 'wa', 'whi', 'yourselv'] not in
stop_words.
  warnings.warn('Your stop_words may be inconsistent with '


LDA Analysis Complete. Top terms extracted:

  RF4_Topic_1 RF4_Topic_2 RF4_Topic_3 RF4_Topic_4 GI5_Topic_1 GI5_Topic_2  \
0        user         thi         thi  blockchain     develop        term
1     develop      commun      commun       secur        user        long
2     network       futur       think      commun  blockchain      commun
3        term        term        like       ensur        fund     strateg
4     support       short        good      govern   ecosystem    prioriti

  GI5_Topic_3 GI5_Topic_4 GI5_Topic_5 V7_Topic_1  ... M15_Topic_6 M15_Topic_7  \
0         thi  blockchain         thi        thi  ...  blockchain        join
1      commun       secur         use     commun  ...        enabl  cryptocurr
2       great       creat        long     vision  ...         real     exchang
3        team      system        make      futur  ...        token         set
4       futur       sybil        need       veri  ...    ownership        earn

  M15_Topic_8 M15_Topic_9 M15_Topic_10 M15_Topic_11 M15_Topic_12 M15_Topic_13  \
0  blockchain         thi         grow       commun       commun      mission
1       chain        best      continu         vote      mission      decentr
2       secur        veri           ye      mission         valu        world
3    industri       great          put       reward        decis       vision
4      enhanc        good        choic       propos        ensur        futur

  M15_Topic_14 M15_Topic_15
0         valu      mission
1         term      network
2         long          use
3      success        peopl
4       vision         make

[5 rows x 52 columns]
```

By looking at the word clusters for each topic (e.g., `RF4_Topic_1`), we can assign a human-readable label. For instance, a topic with words like "communiti," "blockchain," "develop," "secur," and "support" is possibly about **Governance**.

5 Synthesizing Insights with the OpenAI API

We now have structured data: n-grams and topics. But we still need to produce a high-level summary. This is where an LLM like GPT-4o mini shines. We will ask the AI to act as an analyst, providing it with our topic keywords and the raw submissions, and instructing it to generate a synthesized report.

Step 1: Prepare the Prompt

The key to getting a good result from an LLM is a well-structured prompt. Our prompt will:

1. **Assign a role**: "You are an expert blockchain community analyst."
2. **Provide context**: Give it the top keywords from our LDA model.
3. **Provide the data**: Give it the raw text submissions.
4. **Give clear instructions**: Ask it to identify themes and find standout submissions.
5. **Specify the output format**: Request clear headings for easy reading.

[7]:
```
# --- OpenAI Summarization ---

# Let's focus on the "Reduce Friction" contest (RF4)
slug_to_summarize = 'RF4'
all_submissions = []

# Extract top words for the chosen contest
try:
    rf4_cols = [col for col in results.columns if col.
 ↪startswith(f"{slug_to_summarize}_Topic_")]
    top_words_series = results[rf4_cols].stack().unique()
    top_words = ', '.join(top_words_series)

    # Load all submission text files for the chosen contest
    contest_folder = os.path.join(output_dir, slug_to_summarize)
    for filename in os.listdir(contest_folder):
        if filename.endswith('.txt'):
            with open(os.path.join(contest_folder, filename), 'r',␣
 ↪encoding='utf-8') as f:
                all_submissions.append(f.read())
    submissions_text = '\n'.join(all_submissions)
    submissions_text = submissions_text[:8000] # Truncate to avoid exceeding␣
 ↪token limits

    # Craft the prompt
    prompt = f"""
```

```
    You are an expert blockchain community analyst. Given the following user␣
↪submissions and most common keywords, create a detailed summary for each of␣
↪these categories:
    1. Reduce Friction (overall)
    2. Growth and Innovation
    3. Vision
    4. Mission

    For each category, do the following:
    - Identify and describe the main themes, concerns, and opportunities found␣
↪in the submissions.
    - Provide at least 2-3 representative or standout submissions (either as␣
↪direct quotes or short paraphrases) that best capture the spirit or key␣
↪insights of that category.
    - Where relevant, highlight points of consensus, strong opinions, or␣
↪recurring challenges and suggestions.

    Most common keywords:
    {top_words}

    User submissions:
    {submissions_text}

    Format your output with clear headings for each category, and organize␣
↪within each section as:
    - Themes:
      [Detailed synthesis]
    - Standout submissions:
      - [Quote or paraphrase 1]
      - [Quote or paraphrase 2]
      - [Quote or paraphrase 3]

    If a submission is relevant to more than one category, you may include it in␣
↪multiple sections.
    """
except (NameError, FileNotFoundError) as e:
    print(f"Could not prepare prompt. Did the LDA analysis for␣
↪'{slug_to_summarize}' run successfully?")
    prompt = "" # Set prompt to empty to prevent API call
```

Step 2: Make the API Call

Next, we send our prompt to the OpenAI API using the `requests` package. For this exercise, you will need an API key from OpenAI.

Important: Never hardcode your API key directly in your script. For ease of use, load it from a separate, secure file (that you don't share or commit to version control). For a more advanced secure approach, store your API key as a system environment variable.

[ ]:
```
# --- Call OpenAI API ---
api_key = None
try:
    # Load your API key securely from a file (e.g., a file named 'openai_key.
 ↪txt' in a parent directory)
    with open('../../openai_key.txt', 'r') as f:
        api_key = f.read().strip()
except FileNotFoundError:
    print("OpenAI API key file not found. Please create a file named 'openai_key.
 ↪txt' in the parent directory.")

output_box_style = """
<style>
.output-box {
  border: 2px solid #18bc9c;
  background: #eafaf1;
  padding: 12px;
  margin: 1em 0;
  border-radius: 6px;
  font-size: 1.0em;
  font-family: "Fira Mono", "Menlo", "Monaco", "Consolas", monospace;
  white-space: pre-wrap;
}
</style>
"""

if api_key and prompt:
    headers = {
        'Authorization': f'Bearer {api_key}',
        'Content-Type': 'application/json'
    }
    payload = {
        'model': 'gpt-4o-mini',
        'messages': [{'role': 'user', 'content': prompt}],
        'temperature': 0.5,
        'max_tokens': 1500
    }

    try:
        response = requests.post('https://api.openai.com/v1/chat/completions',␣
 ↪headers=headers, json=payload)
        response.raise_for_status()  # Raise an exception for bad status codes␣
 ↪(4xx or 5xx)

        result = response.json()
        content = result['choices'][0]['message']['content']
```

```
        display(HTML(f"{output_box_style}<div class='output-box'>{content}</
↪div>"))

    except requests.exceptions.RequestException as e:
        error_content = e.response.text if e.response else str(e)
        display(HTML(f"{output_box_style}<div class='output-box'>Error calling␣
↪OpenAI API:<br>{error_content}</div>"))
else:
    display(HTML(f"{output_box_style}<div class='output-box'>Skipping OpenAI API␣
↪call. API key or prompt not available.</div>"))
```

1. Reduce Friction (Overall)

Themes

The submissions highlight significant concerns regarding user experience, transaction speed, and security on the Arbitrum network. A recurring theme is the need to minimize barriers for users, particularly in terms of transaction fees and latency. Many contributors emphasize that reducing friction will not only enhance user accessibility but also attract a broader user base to the platform. The importance of fostering a supportive environment for developers is also noted, as this can lead to innovative solutions that further reduce friction for end-users. Security remains a top priority, with suggestions for increased transparency and educational initiatives to help users safeguard their assets.

Standout submissions

- "ArbitrumDAO should prioritize addressing immediate friction, pain points, and barriers to ensure long-term success. Firstly, it is crucial to improve the user experience by minimizing transaction fees and reducing latency on the Arbitrum network."
- "Arbitrum has to decrease fee and increase transaction speed."
- "I suggest having an engagement with the community... to improve protocols that support user accessibility and enhancing the platform's product experiences is of utmost importance."

---

2. Growth and Innovation

Themes

Submissions in this category reflect a strong desire for growth and innovation within the Arbitrum ecosystem. Contributors emphasize the need for a clear roadmap that outlines strategic priorities for development and user engagement. Suggestions include enhancing developer support through grants and education, fostering community-led initiatives, and exploring new applications, particularly in the DeFi space. The idea of leveraging gamification, such as mini-games, to engage users and attract new participants is also mentioned as a unique approach to drive growth.

Standout submissions

- "In short term, ARBDao can encourage the ecosystem to design more mini-games to encourage the users to dive into ARB, as almost all users like mini-games."
- "Arbitrum need to grant more application, especially AA project, cause it will be the next parative for market."
- "Effective Marketing: Develop a comprehensive marketing strategy to reach your target audience... User-Friendly Interface: Ensur."

---

3. Vision

Themes

The vision articulated by users revolves around creating a decentralized, secure, and scalable blockchain ecosystem that prioritizes user experience and community engagement. Many submissions highlight the importance of transparency and governance in shaping the future of Arbitrum. There is a shared belief that by focusing on user needs and addressing current challenges, Arbitrum can position itself as a leader in the blockchain space. Contributors express hope for a future where the platform not only meets but exceeds user expectations through innovative solutions and community involvement.

Standout submissions

- "Short-term strategic priorities are crucial for ArbitrumDAO to navigate this dynamic landscape effectively, achieve immediate objectives, and set the stage for long-term growth and stability."
- "To talk about frictions, Nowadays the security is a topic of high concern for everyone who is related to cryptocurrency."
- "These strategies aim to enhance user engagement and make the platform more accessible, thereby contributing to its overall success in the short term."

---

4. Mission

Themes

The mission articulated by the contributors emphasizes the need for Arbitrum to enhance user accessibility, improve community support, and foster an environment conducive to innovation. Many submissions call for a commitment to security, transparency, and community engagement as foundational elements of the mission. There is a consensus that by prioritizing these aspects, Arbitrum can create a thriving ecosystem that attracts and retains users while supporting developers in building valuable applications.

Standout submissions

- "ArbitrumDAO should prioritize addressing immediate friction, pain points, and barriers to ensure long-term success."

- "Establishing a support department to address initial issues for new users... enhancing the platform's product experiences is of utmost importance."
- "We're all in! We're encouraging and supporting community-led initiatives with resources, guidance, and incentives galore!"

6 Conclusion and Recommendations

By combining traditional NLP techniques with the power of modern LLMs, we have moved from thousands of raw text submissions to a concise, actionable summary of community feedback.

Key Findings

- **Identified Core Themes**: Using LDA, we successfully identified and categorized the main topics of conversation, such as governance, onboarding, and developer tooling.
- **Pinpointed Specific Suggestions**: N-gram analysis helped highlight specific, recurring phrases and proposals.
- **Synthesized Actionable Insights**: The OpenAI API provided a high-quality narrative summary, saving hours of manual reading and interpretation.

Recommendations for Future Analysis

- **Improve Prompting**: To get more structured data from the community, design submission forms with clearer, more focused prompts.
- **Automate Filtering**: Use regular expressions and readability metrics to automatically filter out low-quality or spam submissions.
- **Integrate Voting Data**: Correlate the topics of proposals with their voting outcomes to see which ideas gained the most traction.