# Import necessary libraries

import pandas as pd

import numpy as np

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.cluster import KMeans

import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

# Download necessary files from NLTK:

# punkt -> Tokenization

# stopwords -> Stop words removal

nltk.download("punkt")

nltk.download("stopwords")

# Load the reviews dataset and preview it

reviews = pd.read\_csv("reviews.csv")

reviews.head()

# Step 1: Preprocess the negative reviews

# Filter negative reviews (having a score of 1 or 2)

negative\_reviews\_tmp = reviews[(reviews["score"] == 1) | (reviews["score"] == 2)]["content"]

def preprocess\_text(text):

"""Performs all the required steps in the text preprocessing"""

# Tokenizing the text

tokens = word\_tokenize(text)

# Removing stop words and non-alpha characters

filtered\_tokens = [

token

for token in tokens

if token.isalpha() and token.lower() not in stopwords.words("english")

]

return " ".join(filtered\_tokens)

# Apply the preprocessing function to the negative reviews

negative\_reviews\_cleaned = negative\_reviews\_tmp.apply(preprocess\_text)

# Store the preprocessed negative reviews in a pandas DataFrame

preprocessed\_reviews = pd.DataFrame({"review": negative\_reviews\_cleaned})

preprocessed\_reviews.head()

# Step 2: Vectorize the cleaned negative reviews using TF-IDF

# Vectorize the cleaned reviews using TF-IDF

vectorizer = TfidfVectorizer()

tfidf\_matrix = vectorizer.fit\_transform(preprocessed\_reviews["review"])

# Step 3: Apply K-means clustering to tfidf\_matrix

# Apply K-means clustering (store the model as clust\_kmeans)

clust\_kmeans = KMeans(n\_clusters=5, random\_state=500)

pred\_labels = clust\_kmeans.fit\_predict(tfidf\_matrix)

# Store the predicted labels in a list variable called categories

categories = pred\_labels.tolist()

preprocessed\_reviews["category"] = categories

# Step 4: For each unique cluster label, find the most frequent term

# Get the feature names (terms) from the vectorizer

terms = vectorizer.get\_feature\_names\_out()

# List to save the top term for each cluster

topic\_terms\_list = []

for cluster in range(clust\_kmeans.n\_clusters):

# Get indices of reviews in the current cluster

cluster\_indices = [i for i, label in enumerate(categories) if label == cluster]

# Sum the tf-idf scores for each term in the cluster

cluster\_tfidf\_sum = tfidf\_matrix[cluster\_indices].sum(axis=0)

cluster\_term\_freq = np.asarray(cluster\_tfidf\_sum).ravel()

# Get the top term and its frequencies

top\_term\_index = cluster\_term\_freq.argsort()[::-1][0]

# Append rows to the topic\_terms DataFrame with three fields:

# - category: label / cluster assigned from K-means

# - term: the identified top term

# - frequency: term's weight for the category

topic\_terms\_list.append(

{

"category": cluster,

"term": terms[top\_term\_index],

"frequency": cluster\_term\_freq[top\_term\_index],

}

)

# Pandas DataFrame to store results from this step

topic\_terms = pd.DataFrame(topic\_terms\_list)

# Output the final result

print(topic\_terms)