import pandas as pd

Imports the Pandas library, which is used for data manipulation and analysis.

from serpapi import GoogleSearch

Imports the GoogleSearch class from the serpapi library, which allows for programmatic access to Google search results.

from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer

Imports the SentimentIntensityAnalyzer from the vaderSentiment library, which is used for sentiment analysis.

# Initialize the VADER sentiment analyzer

analyzer = SentimentIntensityAnalyzer()

Creates an instance of the sentiment analyzer to analyze text sentiment.

# Function to calculate sentiment score

def calculate\_sentiment(text):

return analyzer.polarity\_scores(text)['compound']

Defines a function that takes text as input and returns the compound sentiment score using the VADER analyzer.

# Function to classify risk based on text and sentiment score

def identify\_risk(text, sentiment\_score):

risk\_keywords = ['risk', 'danger', 'uncertain', 'hazard', 'threat']

Defines a function to identify whether a paper mentions risks. It initializes a list of keywords associated with risk.

if any(word in text.lower() for word in risk\_keywords) or sentiment\_score <= -0.3:

return 'Risk'

Checks if any risk keywords are present in the text or if the sentiment score is less than or equal to -0.3. If either condition is met, it classifies the paper as 'Risk'.

return 'No Risk'

If neither condition is met, it classifies the paper as 'No Risk'.

# Google Scholar API search parameters

search\_params = {

"engine": "google\_scholar",

"hl": "en",

"num": 10,

"start": 0,

"api\_key": "your\_api\_key\_here",

}

Sets the parameters for the Google Scholar search, including the search engine, language, number of results per page, starting index, and API key.

# List of topics for collection

topics = ["biology", "chemistry", "physics", "information technology"]

Defines a list of research topics for which papers will be searched.

# Initialize lists to store paper information

total\_pages = 3

Initializes a variable to specify the total number of pages to retrieve for each topic.

# Loop through each topic and collect data

for topic in topics:

Starts a loop to iterate through each topic in the topics list.

search\_params["q"] = topic

paper\_titles, paper\_abstracts, citation\_counts, pub\_dates = [], [], [], []

Updates the search query to the current topic and initializes lists to store titles, abstracts, citation counts, and publication dates.

for pg in range(total\_pages):

Starts another loop to iterate through the specified number of pages.

search\_params["start"] = pg \* search\_params["num"]

search = GoogleSearch(search\_params)

search\_results = search.get\_dict()

scholarly\_papers = search\_results.get("organic\_results", [])

Calculates the starting index for the search results based on the current page number, performs the search using GoogleSearch, and retrieves the results.

# Parse paper data

for paper in scholarly\_papers:

Starts a loop to parse the data for each paper in the search results.

paper\_titles.append(paper.get('title', ''))

paper\_abstracts.append(paper.get('snippet', ''))

citation\_counts.append(int(paper.get('inline\_links', {}).get('cited\_by', {}).get('total', 0)))

pub\_dates.append(paper.get('publication\_info', {}).get('summary', ''))

Extracts and appends the title, abstract, citation count, and publication date of each paper to their respective lists.

# Create DataFrame for the topic

topic\_df = pd.DataFrame({

'Title': paper\_titles,

'Abstract': paper\_abstracts,

'Citation Count': citation\_counts,

'Publication Date': pub\_dates

})

Creates a Pandas DataFrame from the collected paper data.

# Apply VADER sentiment analysis to the abstracts and classify risk

topic\_df['VADER Sentiment Score'] = topic\_df['Abstract'].apply(calculate\_sentiment)

Applies the calculate\_sentiment function to the 'Abstract' column to compute sentiment scores and adds these scores to a new column.

topic\_df['Risk'] = topic\_df.apply(lambda row: identify\_risk(row['Abstract'], row['VADER Sentiment Score']), axis=1)

Uses the identify\_risk function to classify each paper based on its abstract and sentiment score, adding the result to a new 'Risk' column.

# Save the results to CSV

topic\_df.to\_csv(f'{topic}\_papers.csv', index=False)

Saves the DataFrame to a CSV file named after the topic.

print(f"Data collection for {topic} completed. Saved to {topic}\_papers.csv.")

Prints a message indicating that data collection for the current topic is complete.

# Display a few examples with Risk and No Risk classification

for risk\_status in ['Risk', 'No Risk']:

Starts a loop to display examples of papers classified as either 'Risk' or 'No Risk'.

filtered\_df = topic\_df[topic\_df['Risk'] == risk\_status]

sample\_size = min(5, len(filtered\_df))

Filters the DataFrame for papers with the current risk status and determines the sample size to display.

if sample\_size > 0:

examples = filtered\_df.sample(n=sample\_size, random\_state=42)

print(f"\nExamples of {risk\_status} papers for {topic}:")

print(examples[['Title', 'Abstract', 'Risk', 'VADER Sentiment Score']])

If there are examples available, randomly selects a sample of papers and prints their titles, abstracts, risk classification, and sentiment scores.

else:

print(f"\nNo examples of {risk\_status} papers available for {topic}.")

# new version

1. First, we added a dictionary of emotion categories:

EMOTION\_CATEGORIES = {

'Joy': ['happy', 'excited', 'delighted', 'pleased', 'optimistic', 'positive', 'successful', 'breakthrough'],

'Confidence': ['proven', 'demonstrated', 'established', 'confirmed', 'verified', 'validated', 'certain'],

*# ... other emotions*

}

This replaces the original risk\_keywords list. Instead of looking for risk-related words, we're now looking for emotion-indicating words in academic writing.

1. We replaced calculate\_sentiment function with a more comprehensive emotion analysis:

python

Copy

def calculate\_emotion\_scores(text):

text\_lower = text.lower()

emotion\_scores = {}

*# Count occurrences of emotion keywords*

for emotion, keywords in EMOTION\_CATEGORIES.items():

score = sum(1 for keyword in keywords if keyword in text\_lower)

emotion\_scores[emotion] = score

*# Get the VADER sentiment score as well*

sentiment\_score = analyzer.polarity\_scores(text)['compound']

emotion\_scores['Sentiment\_Score'] = sentiment\_score

return emotion\_scores

This function:

Takes a text input converts it to lowercase for matching counts how many times each emotion's keywords appear keeps the vader sentiment score you had originally returns a dictionary with scores for each emotion

1. Added a new function to identify primary emotions:

def identify\_primary\_emotions(emotion\_scores, threshold=0):

scores\_without\_sentiment = {k: v for k, v in emotion\_scores.items() if k != 'Sentiment\_Score'}

primary\_emotions = [emotion for emotion, score in scores\_without\_sentiment.items() if score > threshold]

if not primary\_emotions:

return ['Neutral']

return primary\_emotions

This replaces identify\_risk function and takes the emotion scores dictionary removes the sentiment score from consideration returns a list of emotions that score above zero returns 'neutral' if no strong emotions are detected

1. In your main data collection loop, we modified how the analysis is applied:

*# Original version:*

topic\_df['VADER Sentiment Score'] = topic\_df['Abstract'].apply(calculate\_sentiment)

topic\_df['Risk'] = topic\_df.apply(lambda row: identify\_risk(row['Abstract'], row['VADER Sentiment Score']), axis=1)

*# New version:*

emotion\_analysis = topic\_df['Abstract'].apply(calculate\_emotion\_scores)

*# Add emotion scores as separate columns*

for emotion in EMOTION\_CATEGORIES.keys():

topic\_df[f'{emotion}\_Score'] = emotion\_analysis.apply(lambda x: x[emotion])

*# Add VADER sentiment score*

topic\_df['Sentiment\_Score'] = emotion\_analysis.apply(lambda x: x['Sentiment\_Score'])

*# Add primary emotions*

topic\_df['Primary\_Emotions'] = emotion\_analysis.apply(identify\_primary\_emotions)