prompt\_template defines the format of the input.

llm is the language model that processes the input here is langchain\_openai

content\_chain is the RunnableSequence that links the prompt template and the language model.

1. **Importing Libraries**:

**Python**

import os

import csv

import time

from langchain.prompts import PromptTemplate

from langchain\_core.runnables.base import RunnableSequence

from langchain\_openai import OpenAI

from dotenv import load\_dotenv

* + os: Provides functions to interact with the operating system.
  + csv: Used for reading and writing CSV files.
  + time: Provides time-related functions.
  + langchain.prompts.PromptTemplate: Used to create prompt templates for language models.
  + langchain\_core.runnables.base.RunnableSequence: Used to create sequences of runnables.
  + langchain\_openai.OpenAI: Used to interact with OpenAI’s API.
  + dotenv.load\_dotenv: Loads environment variables from a .env file.

1. **Loading Environment Variables**:

**Python**

load\_dotenv()

* + Loads environment variables from a .env file into the environment.

1. **Accessing and Printing the API Key**:

**Python**

api\_key = os.getenv("OPENAI\_API\_KEY")

if not api\_key:

raise ValueError("OpenAI API key not found in environment variables")

print(api\_key)

* + Retrieves the OpenAI API key from the environment variables.
  + Raises an error if the API key is not found.
  + Prints the API key.

1. **Initializing the OpenAI Client**:

**Python**

llm = OpenAI(temperature=0, openai\_api\_key=api\_key)

* + Initializes the OpenAI client with the API key and sets the temperature to 0 (deterministic responses).

1. **Defining the Prompt Template for Sentiment Analysis**:

**Python**

template = """You are a sentiment analyst. Analyze the following statement and respond with either "positive" or "negative".

Statement: {content}

YOUR RESPONSE:

"""

prompt\_template = PromptTemplate(input\_variables=["content"], template=template)

* + Creates a prompt template for sentiment analysis, where {content} will be replaced with the actual content to analyze.

1. **Creating the RunnableSequence for Sentiment Analysis**:

**Python**

content\_chain = prompt\_template | llm

review = content\_chain.invoke({"content": content}).strip('"')

* + Combines the prompt template and the OpenAI client into a sequence that can be run to perform sentiment analysis.

A RunnableSequence in LangChain is a sequence of runnable components where the output of each component serves as the input for the next. This sequence is crucial for creating complex chains of operations, such as those needed for tasks like sentiment analysis. The primary purpose of a RunnableSequence is to streamline the execution of multiple tasks in a defined order. It ensures that each step in the sequence receives the necessary input from the previous step, making it easier to manage and execute complex workflows.

1. **Defining File Paths**:

**Python**

input\_csv\_file\_path = 'tweets-labels.csv'

output\_csv\_file\_path = 'output.csv'

* + Specifies the paths to the input and output CSV files.

1. **Function to Process a Batch of Rows**:

**Python**

def process\_batch(rows, csvwriter):

for row in rows:

content = row[0] # Assuming the content is in the first column

try:

review = content\_chain.invoke({"content": content}).strip('"')

csvwriter.writerow([review, row[1], row[0]])

print(review, row[1], row[0])

except Exception as e:

print(f"Error processing row: {row}, Error: {e}")

outfile.flush()

* + Processes a batch of rows, performs sentiment analysis on each row, and writes the results to the output CSV file.

1. **Function to Process Data with a Delay Every 1500 Rows**:

**Python**

def process\_data(rows):

for i, row in enumerate(rows):

if (i + 1) % 1500 == 0:

time.sleep(61) # Wait for 1 minute

* + Adds a delay every 1500 rows to respect rate limits.

1. **Reading the Input CSV File and Processing Each Row in Batches**:

**Python**

batch\_size = 1000 # Adjust the batch size as needed

with open(input\_csv\_file\_path, mode='r', newline='', encoding='utf-8') as infile, \

open(output\_csv\_file\_path, mode='w', newline='', encoding='utf-8') as outfile:

csvreader = csv.reader(infile)

csvwriter = csv.writer(outfile)

csvwriter.writerow(['Review', 'Original Content', 'Label'])

batch = []

for row in csvreader:

batch.append(row)

if len(batch) >= batch\_size:

process\_batch(batch, csvwriter)

batch = []

time.sleep(70) # Add delay to respect rate limits

if batch:

process\_batch(batch, csvwriter)

* + Reads the input CSV file and processes each row in batches.
  + Writes the results to the output CSV file.

Explain evaluate-with-lexicon.py

1. **Importing the CSV Module**:

**Python**

import csv

* + This imports the csv module, which provides functionality to read from and write to CSV files.

1. **Function to Read a List from a CSV File**:

**Python**

def read\_list\_from\_csv(file\_path):

with open(file\_path, 'r', newline='', encoding='latin-1') as file:

reader = csv.reader(file)

return [row[0].strip() for row in reader] # Assuming the list is in the first column

* + This function reads a CSV file and returns a list of items from the first column.
  + file\_path: The path to the CSV file.
  + Opens the file in read mode with latin-1 encoding.
  + Uses csv.reader to read the file.
  + Returns a list of stripped strings from the first column.

1. **Function to Check Items in the List Against Content**:

**Python**

def check\_items\_in\_list(content\_file, list\_file):

items\_list = read\_list\_from\_csv(list\_file)

print(items\_list)

found\_items = []

not\_found\_items = []

with open(content\_file, 'r', newline='', encoding='latin-1') as file:

reader = csv.reader(file)

for row in reader:

if len(row) > 2:

third\_column\_item = row[2].strip().strip('"')

words = third\_column\_item.split()

found = False

for word in words:

word = word.replace("!", "")

if word in items\_list:

found = True

found\_items.append([word, row])

break

if not found:

not\_found\_items.append(row)

with open('found\_items-in-array-4.csv', 'w', newline='', encoding='latin-1') as file:

writer = csv.writer(file)

writer.writerows(found\_items)

with open('not\_found\_items-in-array-4.csv', 'w', newline='', encoding='latin-1') as file:

writer = csv.writer(file)

writer.writerows(not\_found\_items)

* + This function checks if words from the third column of a content CSV file are in a list read from another CSV file.
  + content\_file: The path to the CSV file containing the content.
  + list\_file: The path to the CSV file containing the list.
  + Reads the list from list\_file using read\_list\_from\_csv.
  + Initializes found\_items and not\_found\_items lists.
  + Opens the content\_file and reads it using csv.reader.
  + For each row, checks if the third column contains any word from the list.
  + If a word is found, adds the word and row to found\_items; otherwise, adds the row to not\_found\_items.
  + Writes found\_items and not\_found\_items to separate CSV files.

1. **Example Usage**:

**Python**

content\_file\_path = 'array\_four.csv' # Path to the file containing the content

list\_file\_path = 'full\_hatebase\_element2.csv' # Path to the file containing the list

check\_items\_in\_list(content\_file\_path, list\_file\_path)

AI-generated code. Review and use carefully. [More info on FAQ](https://www.bing.com/new#faq).

* + Specifies the paths to the content and list CSV files.
  + Calls check\_items\_in\_list with these file paths to perform the check.

This code essentially reads a list of items from one CSV file and checks if any of these items appear in the third column of another CSV file. It then writes the results to two new CSV files: one for found items and one for not found items.

1. **Importing the CSV Module**:

**Python**

import csv

* + This imports the csv module, which provides functionality to read from and write to CSV files.

1. **Initializing Arrays**:

**Python**

array\_one = [] # Positive reviews with Original content 1

array\_two = [] # Negative reviews with Original content 0

array\_three = [] # Positive reviews with Original content 0

array\_four = [] # Negative reviews with Original content 1

unmatched\_condition = [] # Catch errors

* + These arrays will store rows based on specific conditions related to review sentiment and original content labels.

1. **Reading the CSV File**:

**Python**

with open('output.csv', mode='r', encoding='utf-8') as csvfile:

csv\_reader = csv.reader(csvfile, delimiter=',')

next(csv\_reader) # Skip header row

total\_rows = 0

for row in csv\_reader:

if len(row) < 3: # Ensure the row has at least 3 columns

continue

total\_rows += 1

review\_sentiment = row[0].strip().strip('"').lower() # Normalize case and remove newline characters

# Convert "neutral" to "positive"

if review\_sentiment in ["positive", "neutral"]:

review\_sentiment = "positive"

label = int(row[1])

if (review\_sentiment == "positive" and label == 1):

array\_one.append(row)

elif (review\_sentiment == "negative" and label == 0):

array\_two.append(row)

elif (review\_sentiment == "positive" and label == 0):

array\_three.append(row)

elif (review\_sentiment == "negative" and label == 1):

array\_four.append(row)

else:

original\_content = row[0]

print(f"Raw row data: {row}")

print(f"Unmatched condition: '{review\_sentiment}', {label}, original content, {original\_content} ")

unmatched\_condition.append(row)

* + Opens the CSV file output.csv in read mode.
  + Skips the header row.
  + Iterates through each row, normalizes the review sentiment, and categorizes the rows into different arrays based on the conditions.

1. **Calculating Percentages**:

**Python**

if total\_rows > 0:

percent\_array\_one = (len(array\_one) / total\_rows) \* 100

percent\_array\_two = (len(array\_two) / total\_rows) \* 100

percent\_array\_three = (len(array\_three) / total\_rows) \* 100

percent\_array\_four = (len(array\_four) / total\_rows) \* 100

percent\_unmatched\_condition = (len(unmatched\_condition) / total\_rows) \* 100

else:

percent\_array\_one = percent\_array\_two = percent\_array\_three = percent\_array\_four = 0

* + Calculates the percentage of rows in each array relative to the total number of rows.

1. **Writing Results to a File**:

**Python**

with open('sorted\_reviews.csv', mode='w', encoding='utf-8', newline='') as outfile:

csv\_writer = csv.writer(outfile, delimiter=',')

csv\_writer.writerow(['Review Sentiment', 'Original Content', 'Text'])

csv\_writer.writerow(['Array One (Positive, Original Content 1)'])

csv\_writer.writerows(array\_one)

csv\_writer.writerow(['Array Two (Negative, Original Content 0)'])

csv\_writer.writerows(array\_two)

csv\_writer.writerow(['Array Three (Positive, Original Content 0)'])

csv\_writer.writerows(array\_three)

csv\_writer.writerow(['Array Four (Negative, Original Content 1)'])

csv\_writer.writerows(array\_four)

* + Writes the categorized rows to a new CSV file sorted\_reviews.csv.

1. **Writing Percentages to Another File**:

**Python**

with open('percentages.csv', mode='w', encoding='utf-8', newline='') as percent\_file:

csv\_writer = csv.writer(percent\_file, delimiter=',')

csv\_writer.writerow(['Array', 'Percentage'])

csv\_writer.writerow(['Array One (Positive, Original Content 1)', f'{percent\_array\_one:.2f}%'])

csv\_writer.writerow(['Array Two (Negative, Original Content 0)', f'{percent\_array\_two:.2f}%'])

csv\_writer.writerow(['Array Three (Positive, Original Content 0)', f'{percent\_array\_three:.2f}%'])

csv\_writer.writerow(['Array Four (Negative, Original Content 1)', f'{percent\_array\_four:.2f}%'])

* + Writes the calculated percentages to a new CSV file percentages.csv.

1. **Printing Percentages**:

**Python**

print(f"Percentage of Array One: {percent\_array\_one:.2f}%")

print(f"Percentage of Array Two: {percent\_array\_two:.2f}%")

print(f"Percentage of Array Three: {percent\_array\_three:.2f}%")

print(f"Percentage of Array Four: {percent\_array\_four:.2f}%")

if len(unmatched\_condition) > 0:

print(f"Percentage of Error Array : {percent\_unmatched\_condition:.2f}%")

* + Prints the percentages of each array to the console.

This code reads a CSV file, categorizes rows based on review sentiment and original content labels, calculates the percentages of each category, and writes the results to new CSV