In [10]:
```
%pip install pandas openai rouge-score tqdm
```

```
Requirement already satisfied: pandas in /opt/anaconda3/lib/python3.11/site-packages (2.1.4)
Requirement already satisfied: openai in /opt/anaconda3/lib/python3.11/site-packages (1.77.0)
Requirement already satisfied: rouge-score in /opt/anaconda3/lib/python3.11/site-packages (0.1.2)
Requirement already satisfied: tqdm in /opt/anaconda3/lib/python3.11/site-packages (4.65.0)
Requirement already satisfied: numpy<2,>=1.23.2 in /opt/anaconda3/lib/python3.11/site-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/anaconda3/lib/python3.11/site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /opt/anaconda3/lib/python3.11/site-packages (from pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in /opt/anaconda3/lib/python3.11/site-packages (from pandas) (2023.3)
Requirement already satisfied: anyio<5,>=3.5.0 in /opt/anaconda3/lib/python3.11/site-packages (from openai) (4.2.0)
Requirement already satisfied: distro<2,>=1.7.0 in /opt/anaconda3/lib/python3.11/site-packages (from openai) (1.8.0)
Requirement already satisfied: httpx<1,>=0.23.0 in /opt/anaconda3/lib/python3.11/site-packages (from openai) (0.28.1)
Requirement already satisfied: jiter<1,>=0.4.0 in /opt/anaconda3/lib/python3.11/site-packages (from openai) (0.9.0)
Requirement already satisfied: pydantic<3,>=1.9.0 in /opt/anaconda3/lib/python3.11/site-packages (from openai) (1.10.12)
Requirement already satisfied: sniffio in /opt/anaconda3/lib/python3.11/site-packages (from openai) (1.3.0)
Requirement already satisfied: typing-extensions<5,>=4.11 in /opt/anaconda3/lib/python3.11/site-packages (from openai) (4.13.2)
Requirement already satisfied: absl-py in /opt/anaconda3/lib/python3.11/site-packages (from rouge-score) (2.1.0)
Requirement already satisfied: nltk in /opt/anaconda3/lib/python3.11/site-packages (from rouge-score) (3.8.1)
Requirement already satisfied: six>=1.14.0 in /opt/anaconda3/lib/python3.11/site-packages (from rouge-score) (1.16.0)
Requirement already satisfied: idna>=2.8 in /opt/anaconda3/lib/python3.11/site-packages (from anyio<5,>=3.5.0->openai) (3.4)
Requirement already satisfied: certifi in /opt/anaconda3/lib/python3.11/site-packages (from httpx<1,>=0.23.0->openai) (2024.2.2)
Requirement already satisfied: httpcore==1.* in /opt/anaconda3/lib/python3.11/site-packages (from httpx<1,>=0.23.0->openai) (1.0.9)
Requirement already satisfied: h11>=0.16 in /opt/anaconda3/lib/python3.11/site-packages (from httpcore==1.*->httpx<1,>=0.23.0->openai) (0.16.0)
Requirement already satisfied: click in /opt/anaconda3/lib/python3.11/site-packages (from nltk->rouge-score) (8.1.7)
Requirement already satisfied: joblib in /opt/anaconda3/lib/python3.11/site-packages (from nltk->rouge-score) (1.2.0)
Requirement already satisfied: regex>=2021.8.3 in /opt/anaconda3/lib/python3.11/site-packages (from nltk->rouge-score) (2023.10.3)
Note: you may need to restart the kernel to use updated packages.
```

In [11]:
```
# Cell 1: Imports and API Key Setup
import pandas as pd
import openai
from rouge_score import rouge_scorer
import time
import os
from tqdm.notebook import tqdm # Use tqdm.notebook for better notebook progress bars
import warnings

# --- OpenAI API Setup ---
# WARNING: If you save this notebook and share it, your API key will be visible.
# It's better to use a .env file (see Option 2) or input() for shared notebooks.
# For personal use where the notebook isn't shared widely, this is convenient.

# --- REPLACE WITH YOUR ACTUAL KEY, ORG ID, AND PROJECT ID ---
api_key_value = "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"
org_id_value = "org-xwnyLWKd2QBruMFmhMFJeSU0"
project_id_value = "proj_TWJIBSkEnUAiZ5WG1ZfVbvFa" # Project ID is less commonly needed for basic API calls

# Set them as environment variables for the current Python process
os.environ["OPENAI_API_KEY"] = api_key_value
os.environ["OPENAI_ORG_ID"] = org_id_value
if project_id_value: # Only set if you have one and it's relevant
    os.environ["OPENAI_PROJECT_ID"] = project_id_value

# Initialize the OpenAI client. It will pick up the environment variables.
try:
    client = openai.OpenAI()
    # Perform a simple test call to verify authentication (optional but good for debugging)
    # client.models.list()
    # print("OpenAI client initialized and authenticated successfully.")
except Exception as e:
    print(f"Error initializing OpenAI client: {e}")
    warnings.warn("OpenAI client might not be properly authenticated. Check your API key and organization ID.")

# --- Configuration ---
CSV_FILE_PATH = 'reddit_advice_dataset.csv' # Make sure this file is in the same directory as your notebook, or provide the full path
OUTPUT_CSV_FILE_PATH = 'reddit_advice_chatgpt_rouge_scores_notebook.csv'

# !!! IMPORTANT: Inspect your CSV and set these column names correctly !!!
PROMPT_COLUMN_NAME = 'question'  # Replace with the actual column name for the user's prompt
HUMAN_ADVICE_COLUMN_NAME = 'suggestion' # Replace with the actual column name for human advice

OPENAI_MODEL = "gpt-4o-mini"
MAX_TOKENS_RESPONSE = 250
TEMPERATURE = 0.7
```

In [12]:
```
# Cell 2: Helper Function for ChatGPT
def get_chatgpt_advice(prompt_text):
    """Generates advice from ChatGPT for a given prompt."""
    try:
        response = client.chat.completions.create( # Use the client initialized in the previous cell
            model=OPENAI_MODEL,
            messages=[
                {"role": "system", "content": "You are a helpful assistant providing advice."},
                {"role": "user", "content": f"Please provide advice for the following situation: {prompt_text}"}
            ],
            max_tokens=MAX_TOKENS_RESPONSE,
            temperature=TEMPERATURE
        )
        return response.choices[0].message.content.strip()
    except Exception as e:
        print(f"Error calling OpenAI API for prompt '{prompt_text[:50]}...': {e}")
        return None
```

In [13]:
```
# Cell 3: Main Processing Logic
def main_notebook_processing():
    # Load the dataset
    try:
        df = pd.read_csv(CSV_FILE_PATH)
    except FileNotFoundError:
        print(f"Error: CSV file not found at {CSV_FILE_PATH}")
        print("Please ensure 'reddit_advice_dataset.csv' is in the same directory as the notebook or provide the full path.")
        return

    print(f"Loaded {len(df)} prompts from {CSV_FILE_PATH}")

    # Initialize ROUGE scorer
    scorer = rouge_scorer.RougeScorer(['rouge1', 'rouge2', 'rougeL'], use_stemmer=True)

    results = []

    print(f"Generating ChatGPT responses and calculating ROUGE scores using {OPENAI_MODEL}...")
    # Use tqdm.notebook for a nice progress bar in the notebook
    for index, row in tqdm(df.iterrows(), total=df.shape[0], desc="Processing Prompts"):
        prompt_text = row[PROMPT_COLUMN_NAME]
        human_advice_text = str(row[HUMAN_ADVICE_COLUMN_NAME])

        if pd.isna(prompt_text) or pd.isna(human_advice_text) or not human_advice_text.strip():
            print(f"Skipping row {index+1} due to missing prompt or human advice.")
            results.append({
                'prompt': prompt_text,
                'human_advice': human_advice_text,
                'chatgpt_advice': "SKIPPED_EMPTY_INPUT",
                'rouge1_f': 0, 'rouge1_p': 0, 'rouge1_r': 0,
                'rouge2_f': 0, 'rougeL_f': 0
            })
            continue

        chatgpt_advice_text = get_chatgpt_advice(prompt_text)

        if chatgpt_advice_text:
            rouge_scores_dict = scorer.score(target=human_advice_text, prediction=chatgpt_advice_text)
            results.append({
                'prompt': prompt_text,
                'human_advice': human_advice_text,
                'chatgpt_advice': chatgpt_advice_text,
                'rouge1_f': rouge_scores_dict['rouge1'].fmeasure,
                'rouge1_p': rouge_scores_dict['rouge1'].precision,
                'rouge1_r': rouge_scores_dict['rouge1'].recall,
                'rouge2_f': rouge_scores_dict['rouge2'].fmeasure,
                'rougeL_f': rouge_scores_dict['rougeL'].fmeasure
            })
        else:
            results.append({
                'prompt': prompt_text,
                'human_advice': human_advice_text,
                'chatgpt_advice': "ERROR_GENERATING_RESPONSE",
                'rouge1_f': 0, 'rouge1_p': 0, 'rouge1_r': 0,
                'rouge2_f': 0, 'rougeL_f': 0
            })

        # time.sleep(0.2) # Optional: slight delay if you encounter rate limits

    results_df = pd.DataFrame(results)
    results_df.to_csv(OUTPUT_CSV_FILE_PATH, index=False)
    print(f"\nResults saved to {OUTPUT_CSV_FILE_PATH}")

    if not results_df.empty:
        # Filter out rows where ROUGE scores might be 0 due to errors/skips for accurate averaging
        valid_scores_df = results_df[results_df['chatgpt_advice'] != "ERROR_GENERATING_RESPONSE"]
        valid_scores_df = valid_scores_df[valid_scores_df['chatgpt_advice'] != "SKIPPED_EMPTY_INPUT"]

        if not valid_scores_df.empty:
            avg_rouge1_f = valid_scores_df['rouge1_f'].mean()
            avg_rouge2_f = valid_scores_df['rouge2_f'].mean()
            avg_rougeL_f = valid_scores_df['rougeL_f'].mean()
            print("\nAverage ROUGE F1-Scores (for successfully processed prompts):")
            print(f"  ROUGE-1: {avg_rouge1_f:.4f}")
            print(f"  ROUGE-2: {avg_rouge2_f:.4f}")
            print(f"  ROUGE-L: {avg_rougeL_f:.4f}")
        else:
            print("\nNo prompts were successfully processed to calculate average ROUGE scores.")

# Call the main processing function
main_notebook_processing()
```

```
Loaded 224 prompts from reddit_advice_dataset.csv
Generating ChatGPT responses and calculating ROUGE scores using gpt-4o-mini...
Processing Prompts:   0%|          | 0/224 [00:00<?, ?it/s]
Results saved to reddit_advice_chatgpt_rouge_scores_notebook.csv

Average ROUGE F1-Scores (for successfully processed prompts):
  ROUGE-1: 0.1849
  ROUGE-2: 0.0249
  ROUGE-L: 0.0975
```