Question 02: Sentiment Analysis and Topic Modeling

Goal: Process a list of social media posts, clean the text, score the sentiment, and find trending negative topics for the brand "GulPhone". **Topics:** Functions, Data Structures (Lists, Dictionaries, Sets), List Comprehensions, `map`, `lambda`.

Provided Data

In [16]:
```
allPosts = [
    {'id': 1, 'text': 'I LOVE the new #GulPhone! Battery life is amazing.'},
    {'id': 2, 'text': 'My #GulPhone is a total disaster. The screen is already broken!'},
    {'id': 3, 'text': 'Worst customer service ever from @GulPhoneSupport. Avoid this.'},
    {'id': 4, 'text': 'The @GulPhoneSupport team was helpful and resolved my issue. Great service!'},
]

PUNCTUATION_CHARS = '!\"#$%&\'()*+,-./:;⟺?@[\\]^_^{|}~'
STOPWORDS_SET = {'i', 'me', 'my', 'a', 'an', 'the', 'is', 'am', 'was', 'and',
                 'but', 'if', 'or', 'to', 'of', 'at', 'by', ' ', 'for', 'with',
                 'this', 'that'}
POSITIVE_WORDS_SET = {'love', 'amazing', 'great', 'helpful', 'resolved'}
NEGATIVE_WORDS_SET = {'disaster', 'broken', 'worst', 'avoid', 'bad'}
```

Tasks

1. **Clean Text:** Write a reusable function `preprocessText(text, punctuationList, stopwordsSet)`.
   - Normalize the text by converting it to **lowercase**.
   - Remove all characters in `punctuationList`.
   - Filter out low-value words (words in `stopwordsSet`).

In [17]:
```
def preprocessText(text, punctuationList, stopwordsSet):
    """
    Normalizes text by converting to lowercase, removing punctuatio
n,
    and filtering out stopwords.

    Args:
        text (str): The input text.
        punctuationList (str): A string of characters to remove.
        stopwordsSet (set): A set of words to filter out.

    Returns:
        list: A list of processed words.
    """
    # Normalize to lowercase
    text = text.lower()

    # Remove punctuation
    text = "".join([char for char in text if char not in punctuation
List])

    # Filter out stopwords and split into words
    words = [word for word in text.split() if word not in stopwordsS
et]

    return words

# Test the function
test_text = "I LOVE the new #GulPhone! Battery life is amazing."
processed_words = preprocessText(test_text, PUNCTUATION_CHARS, STOPW
ORDS_SET)
print(f"Original text: {test_text}")
print(f"Processed words: {processed_words}")
```

```
Original text: I LOVE the new #GulPhone! Battery life is amazing.
Processed words: ['love', 'new', 'gulphone', 'battery', 'life', 'ama
zing']
```

1. **Score Sentiment:** Write a function `analyzePosts(postsList, punctuation, stopwords, positive, negative)` .

   - Use the **`map`** function (with a `lambda`) to efficiently apply `preprocessText` to every post.
   - Score each post: **+1** for each word in `POSITIVE_WORDS_SET` , **-1** for each word in `NEGATIVE_WORDS_SET` .
   - Return a new list of dictionaries, each containing the **original id, text, the `processedText` list, and the final score**.

In [18]:
```
def analyzePosts(postsList, punctuation, stopwords, positive, negative):
    """
    Analyzes a list of posts, preprocesses the text, scores sentiment,
    and returns a new list of dictionaries with the results.

    Args:
        postsList (list): A list of dictionaries, each representing a post.
        punctuation (str): A string of characters to remove during preprocessing.
        stopwords (set): A set of stopwords to filter out during preprocessing.
        positive (set): A set of positive words for sentiment scoring.
        negative (set): A set of negative words for sentiment scoring.

    Returns:
        list: A list of dictionaries with original id, text, processed text, and score.
    """
    def score_sentiment(processed_words, positive_set, negative_set):
        score = 0
        for word in processed_words:
            if word in positive_set:
                score += 1
            elif word in negative_set:
                score -= 1
        return score

    analyzed_posts = list(map(lambda post: {
        'id': post['id'],
        'text': post['text'],
        'processedText': preprocessText(post['text'], punctuation, stopwords),
        'score': score_sentiment(preprocessText(post['text'], punctuation, stopwords), positive, negative)
    }, postsList))

    return analyzed_posts

# Test the function
analyzed_results = analyzePosts(allPosts, PUNCTUATION_CHARS, STOPWORDS_SET, POSITIVE_WORDS_SET, NEGATIVE_WORDS_SET)
display(analyzed_results)
```

```
[{'id': 1,
  'text': 'I LOVE the new #GulPhone! Battery life is amazing.',
  'processedText': ['love', 'new', 'gulphone', 'battery', 'life', 'a
mazing'],
  'score': 2},
 {'id': 2,
  'text': 'My #GulPhone is a total disaster. The screen is already b
roken!',
  'processedText': ['gulphone',
   'total',
   'disaster',
   'screen',
   'already',
   'broken'],
  'score': -2},
 {'id': 3,
  'text': 'Worst customer service ever from @GulPhoneSupport. Avoid
this.',
  'processedText': ['worst',
   'customer',
   'service',
   'ever',
   'from',
   'gulphonesupport',
   'avoid'],
  'score': -2},
 {'id': 4,
  'text': 'The @GulPhoneSupport team was helpful and resolved my iss
ue. Great service!',
  'processedText': ['gulphonesupport',
   'team',
   'helpful',
   'resolved',
   'issue',
   'great',
   'service'],
  'score': 3}]
```

1. **Flag Posts:** Write a function `getFlaggedPosts(scoredPosts, sentimentThreshold=-1)` .

   - Use a **single-line list comprehension** to return a new list containing only the posts at or below the `sentimentThreshold` .

In [19]:
```
def getFlaggedPosts(scoredPosts, sentimentThreshold=-1):
    """
    Filters a list of scored posts to return only those at or below
a given sentiment threshold.

    Args:
        scoredPosts (list): A list of dictionaries, each representin
g a scored post.
        sentimentThreshold (int): The threshold score. Posts with sc
ores at or below this will be flagged.

    Returns:
        list: A list of dictionaries containing only the flagged pos
ts.
    """
    return [post for post in scoredPosts if post['score'] ⩽ sentime
ntThreshold]

# Test the function
flagged_posts = getFlaggedPosts(analyzed_results)
display(flagged_posts)
```

```
[{'id': 2,
  'text': 'My #GulPhone is a total disaster. The screen is already b
roken!',
  'processedText': ['gulphone',
   'total',
   'disaster',
   'screen',
   'already',
   'broken'],
  'score': -2},
 {'id': 3,
  'text': 'Worst customer service ever from @GulPhoneSupport. Avoid
this.',
  'processedText': ['worst',
   'customer',
   'service',
   'ever',
   'from',
   'gulphonesupport',
   'avoid'],
  'score': -2}]
```

1. **Find Topics:** Write a function `findNegativeTopics(flaggedPosts)` .

   - Iterate only through the list of negative posts from Task 3 to find all **hashtags (#) and mentions (@)**.
   - Return a dictionary that counts the **frequency** of each negative topic.

In [21]:
```
import re

def findNegativeTopics(flaggedPosts):
    """
    Finds and counts hashtags and mentions from a list of flagged po
sts
    by examining the original text.

    Args:
        flaggedPosts (list): A list of dictionaries containing flagg
ed posts.

    Returns:
        dict: A dictionary with the frequency of each hashtag and me
ntion.
    """
    negative_topics = {}
    for post in flaggedPosts:
        # Examine the original text to find hashtags and mentions
        words = post['text'].lower().split()
        for word in words:
            if word.startswith('#') or word.startswith('@'):
                # Clean the word to remove trailing punctuation
                cleaned_word = "".join([char for char in word if cha
r.isalnum() or char in ['#', '@']])
                if cleaned_word: # Ensure the cleaned word is not em
pty
                    if cleaned_word in negative_topics:
                        negative_topics[cleaned_word] += 1
                    else:
                        negative_topics[cleaned_word] = 1
    return negative_topics

# Test the function
negative_topics = findNegativeTopics(flagged_posts)
display(negative_topics)
```

```
{'#gulphone': 1, '@gulphonesupport': 1}
```

In [20]:
```
```