```
# create a class for the transcript data
import re
from collections import Counter
from transformers import pipeline
import os



class TranscriptLLM:
    def __init__(self, transcript, sentiment_analyzer, sentiment_analyzer_val):
        self.transcript = transcript
        self.data = None
        self.sentiment_analyzer = sentiment_analyzer
        self.sentiment_analyzer_val = sentiment_analyzer_val

    def determine_call_outcome(self, transcript):
        """
        Infers the call outcome based on common phrases.
        Returns 'Issue Resolved' or 'Follow-up Action Needed'.
        """
            resolved_keywords = ["issue resolved", "problem fixed", "refund processed", "resolved",
"successful"]
        follow_up_keywords = ["call back", "follow-up", "additional information needed", "escalate"]

        # Check for keywords in the transcript
        transcript_lower = transcript #self.transcript.lower()
        if any(phrase in transcript_lower for phrase in resolved_keywords):
```

```
        return "Issue Resolved"
    elif any(phrase in transcript_lower for phrase in follow_up_keywords):
        return "Follow-up Action Needed"
    else:
        return "Follow-up Action Needed"  # Default assumption if uncertain

def analyze_transcript(self, transcript):
    """
    Analyzes the sentiment and outcome of the customer part of a transcript.
    """
    # Extract customer-only lines (assuming "Member" indicates the customer)
    customer_lines = "\n".join([line for line in transcript.splitlines() if line.startswith("Member:")])
    customer_text = re.sub(r"Member:\s*", "", customer_lines)

    # Analyze sentiment
    sentiment_result = self.sentiment_analyzer(customer_text)
    sentiment = sentiment_result[0]['label']

    # Convert sentiment labels to match the evaluation format
    if sentiment == "POSITIVE":
        sentiment = "Positive"
    elif sentiment == "NEGATIVE":
        sentiment = "Negative"
    else:
        sentiment = "Neutral"

    # Determine call outcome
```

```
    outcome = self.determine_call_outcome(customer_text)

    return sentiment, outcome

def test_consistency(self, transcript, iterations=5):
    """
    Runs the model multiple times on the same input to test consistency.
    """
    sentiments = []
    for _ in range(iterations):
        prediction = self.analyze_transcript(transcript)
        sentiment = prediction[0]
        sentiments.append(sentiment)

    # Count occurrences of each label
    label_counts = Counter(sentiments)

    # Most frequent label
    most_common_label, frequency = label_counts.most_common(1)[0]

    consistency_rate = frequency / iterations
    return {
        "most_common_label": most_common_label,
        "consistency_rate": consistency_rate,
        "predictions": sentiments
    }
```

```
    def analyze_transcript_val(self, transcript):
        """
            Analyzes the sentiment and outcome of the customer part of a transcript using a different model.
        """
        # Extract customer-only lines (assuming "Member" indicates the customer)
        customer_lines = "\n".join([line for line in transcript.splitlines() if line.startswith("Member:")])
        customer_text = re.sub(r"Member:\s*", "", customer_lines)

        # Analyze sentiment
        sentiment_result = self.sentiment_analyzer_val(customer_text)
        sentiment = sentiment_result[0]['label']

        # Convert sentiment labels to match the evaluation format
        if sentiment == "LABEL_2":
            sentiment = "Positive"
        elif sentiment == "LABEL_0":
            sentiment = "Negative"
        else:
            sentiment = "Neutral"

        # Determine call outcome
        outcome = self.determine_call_outcome(customer_text)

        return sentiment, outcome

    def evaluate_models(self, transcripts):
```

```
        """
        Evaluates the performance of two sentiment analysis models on the given transcripts.
        """
        predicted_sentiments = []
        predicted_sentiments_val = []
        # model 1
        predicted_sentiment, predicted_outcomes = self.analyze_transcript(self.transcript)
        predicted_sentiments.append(predicted_sentiment)

        # Model 2
        predicted_sentiment_val, predicted_outcomes_val = self.analyze_transcript_val(self.transcript)
        predicted_sentiments_val.append(predicted_sentiment_val)

        # compare agreement between the two models on sentiment
            sentiment_agreement = [pred1 == pred2 for pred1, pred2 in zip(predicted_sentiments, predicted_sentiments_val)]

        # print agreement rate
        agreement_rate = sum(sentiment_agreement) / len(sentiment_agreement)
        return agreement_rate
```