In [ ]:
```
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
import numpy as np
import pandas as pd

file_path = '/mnt/data/impression_300_llm.csv'
data = pd.read_csv(file_path)

data['input_text'] = data['Report Name'].astype(str) + ' ' + data['History'].astype(str) + ' ' + data['Observation'].astype(str)

inputs = data['input_text'].values
targets = data['Impression'].values

max_vocab_size = 10000
tokenizer = Tokenizer(num_words=max_vocab_size, oov_token="<OOV>")
tokenizer.fit_on_texts(inputs)

input_sequences = tokenizer.texts_to_sequences(inputs)
target_sequences = tokenizer.texts_to_sequences(targets)

max_len_input = max([len(seq) for seq in input_sequences])
max_len_target = max([len(seq) for seq in target_sequences])

padded_inputs = pad_sequences(input_sequences, maxlen=max_len_input, padding='post')
padded_targets = pad_sequences(target_sequences, maxlen=max_len_target, padding='post')

train_inputs, test_inputs, train_targets, test_targets = train_test_split(
    padded_inputs, padded_targets, test_size=0.2, random_state=42
)
```

In [ ]:
```
embedding_dim = 128
lstm_units = 128

model = tf.keras.Sequential([
    tf.keras.layers.Embedding(max_vocab_size, embedding_dim, input_length=max_len_input),
    tf.keras.layers.LSTM(lstm_units, return_sequences=True),
    tf.keras.layers.LSTM(lstm_units),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(max_len_target, activation='softmax')
])

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model.summary()
```

In [ ]:
```
epochs = 10
batch_size = 32

history = model.fit(
    train_inputs, np.expand_dims(train_targets, -1),
    epochs=epochs,
    batch_size=batch_size,
    validation_split=0.2
)

test_loss, test_accuracy = model.evaluate(test_inputs, np.expand_dims(test_targets, -1))
print(f"Test Accuracy: {test_accuracy}")
```

In [ ]:
```
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import nltk

nltk.download('stopwords')
nltk.download('punkt')

stop_words = set(stopwords.words('english'))
ps = PorterStemmer()

def preprocess_text(text):
    words = text.lower().split()
    words = [ps.stem(word) for word in words if word not in stop_words]
    return " ".join(words)

data['processed_text'] = data['Impression'].apply(preprocess_text)

vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(data['processed_text'])

similarity_matrix = cosine_similarity(tfidf_matrix)

top_100_pairs = np.dstack(np.unravel_index(np.argsort(similarity_matrix.ravel())[-200:], similarity_matrix.shape))[-100:]
```

In [ ]:
```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.Graph()

for i, j in top_100_pairs:
    word1 = vectorizer.get_feature_names_out()[i]
    word2 = vectorizer.get_feature_names_out()[j]
    G.add_edge(word1, word2)

plt.figure(figsize=(10, 10))
nx.draw(G, with_labels=True, node_color="lightblue", node_size=3000, font_size=12)
plt.show()
```