```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional
from tensorflow.keras.callbacks import EarlyStopping
import re
import string
import nltk
from nltk.corpus import stopwords

# Set random seeds for reproducibility
np.random.seed(42)
tf.random.set_seed(42)

# Download necessary NLTK data
try:
    nltk.data.find('corpora/stopwords')
except LookupError:
    print("Downloading NLTK stopwords...")
    nltk.download('stopwords')

# a) Select and load a suitable dataset
print("Loading dataset...")

# Load the IMDB Movie Reviews dataset from Keras
from tensorflow.keras.datasets import imdb

# Load the data with the top 10,000 words
max_words = 10000
(X_train_raw, y_train), (X_test_raw, y_test) = imdb.load_data(num_words=max_words)

# Get the word index mapping
word_index = imdb.get_word_index()

# Reverse the word index to get words
reverse_word_index = {value: key for key, value in word_index.items()}

# Function to convert sequences back to text
def seq_to_text(sequence):
    # Index offset is 3 because 0 = padding, 1 = start, 2 = unknown
    return ' '.join([reverse_word_index.get(i-3, '?') for i in sequence if i > 3])

# Convert some examples back to text for inspection
train_reviews = [seq_to_text(sequence) for sequence in X_train_raw[:5]]
print("\nSample reviews from the dataset:")
for i, review in enumerate(train_reviews):
    print(f"Review {i+1} (Sentiment: {'Positive' if y_train[i] == 1 else 'Negative'}):")
    print(review[:200] + "...")  # Print just the beginning of the review

# Display dataset information
print(f"\nTraining dataset size: {len(X_train_raw)} reviews")
print(f"Testing dataset size: {len(X_test_raw)} reviews")

# Display class distribution
train_sentiment_counts = np.bincount(y_train)
test_sentiment_counts = np.bincount(y_test)

print("\nClass distribution in training data:")
print(f"Negative: {train_sentiment_counts[0]} ({train_sentiment_counts[0]/len(y_train)*100:.1f}%)")
print(f"Positive: {train_sentiment_counts[1]} ({train_sentiment_counts[1]/len(y_train)*100:.1f}%)")

# Visualize class distribution
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
sns.countplot(x=y_train)
plt.title('Sentiment Distribution (Training Data)')
plt.xlabel('Sentiment (0: Negative, 1: Positive)')
plt.xticks([0, 1], ['Negative', 'Positive'])

plt.subplot(1, 2, 2)
sns.countplot(x=y_test)
plt.title('Sentiment Distribution (Test Data)')
plt.xlabel('Sentiment (0: Negative, 1: Positive)')
```

```
plt.xticks([0, 1], ['Negative', 'Positive'])

plt.tight_layout()
plt.savefig('sentiment_distribution.png')
plt.close()

# Data preprocessing
print("\nPreprocessing data...")

# Define the max sequence length
max_len = 200  # Maximum sequence length

# Pad the sequences
X_train = pad_sequences(X_train_raw, maxlen=max_len, padding='post', truncating='post')
X_test = pad_sequences(X_test_raw, maxlen=max_len, padding='post', truncating='post')

print(f"Training data shape after padding: {X_train.shape}")
print(f"Testing data shape after padding: {X_test.shape}")

# b) Apply RNN variant (LSTM) for prediction
print("\nBuilding LSTM model...")

# Define the model
def create_lstm_model(vocab_size, embedding_dim=128, lstm_units=64):
    model = Sequential([
        # Embedding layer
        Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=max_len),

        # Bidirectional LSTM layer
        Bidirectional(LSTM(lstm_units, return_sequences=True)),
        Dropout(0.3),

        # Second LSTM layer
        Bidirectional(LSTM(lstm_units)),
        Dropout(0.3),

        # Output layer
        Dense(1, activation='sigmoid')
    ])

    # Compile the model
    model.compile(
        loss='binary_crossentropy',
        optimizer='adam',
        metrics=['accuracy']
    )

    return model

# Create and display the model
vocab_size = max_words + 1  # +1 for the padding token
model = create_lstm_model(vocab_size)
model.summary()

# Train the model
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=3,
    restore_best_weights=True
)

print("\nTraining the model...")
history = model.fit(
    X_train, y_train,
    epochs=10,
    batch_size=64,
    validation_split=0.2,
    callbacks=[early_stopping],
    verbose=1
)

# Plot training history
plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
```

```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

plt.tight_layout()
plt.savefig('training_history.png')
plt.close()

# Evaluate the model
print("\nEvaluating the model...")
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")

# Make predictions
y_pred_proba = model.predict(X_test)
y_pred = (y_pred_proba > 0.5).astype(int)

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Negative', 'Positive'],
            yticklabels=['Negative', 'Positive'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.savefig('confusion_matrix.png')
plt.close()

# Display classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred, target_names=['Negative', 'Positive']))

# Function to predict sentiment on new texts
def predict_sentiment_raw(text, model, max_len=200):
    # Get the word index
    word_index = imdb.get_word_index()

    # Preprocess the text
    text = text.lower()

    # Tokenize
    words = text.split()

    # Convert to sequence
    sequence = []
    for word in words:
        # Add 3 because 0 = padding, 1 = start, 2 = unknown
        idx = word_index.get(word, 0) + 3
        if idx < max_words + 3:  # Only include words in our vocabulary
            sequence.append(idx)

    # Pad the sequence
    padded = pad_sequences([sequence], maxlen=max_len, padding='post', truncating='post')

    # Make prediction
    prediction = model.predict(padded)[0][0]

    return {
        'review': text,
        'sentiment_score': float(prediction),
        'sentiment': 'Positive' if prediction > 0.5 else 'Negative'
    }

# Test the model with a few sample reviews
sample_reviews = [
    "This movie was fantastic! The acting was superb and the plot kept me engaged throughout.",
    "Terrible film. Waste of time and money. The plot made no sense and the acting was wooden.",
    "It was an okay movie. Not great, not terrible, just average entertainment.",
    "I was pleasantly surprised by this film. The reviews weren't great but I really enjoyed it!"
]

print("\nPredictions on sample reviews:")
for review in sample_reviews:
    result = predict_sentiment_raw(review, model)
    print(f"Review: '{review[:50]}...'")
    print(f"Sentiment: {result['sentiment']} (Score: {result['sentiment_score']:.4f})")
    print()
```

```
# Visualize some examples where predictions match and differ from actual labels
correct_predictions = np.where(y_pred == y_test)[0]
incorrect_predictions = np.where(y_pred != y_test)[0]

if len(correct_predictions) > 0 and len(incorrect_predictions) > 0:
    print("\nAnalyzing correct and incorrect predictions:")

    # Sample reviews with correct predictions
    sampled_correct = np.random.choice(correct_predictions, min(3, len(correct_predictions)), replace=False)

    print("\nExamples of CORRECT predictions:")
    for idx in sampled_correct:
        review_text = seq_to_text(X_test_raw[idx])
        sentiment = "Positive" if y_test[idx] == 1 else "Negative"
        print(f"Review: '{review_text[:100]}...'")
        print(f"True sentiment: {sentiment}")
        print(f"Predicted sentiment: {sentiment} (Score: {y_pred_proba[idx][0]:.4f})")
        print()

    # Sample reviews with incorrect predictions
    sampled_incorrect = np.random.choice(incorrect_predictions, min(3, len(incorrect_predictions)), replace=False)

    print("\nExamples of INCORRECT predictions:")
    for idx in sampled_incorrect:
        review_text = seq_to_text(X_test_raw[idx])
        true_sentiment = "Positive" if y_test[idx] == 1 else "Negative"
        pred_sentiment = "Positive" if y_pred[idx] == 1 else "Negative"
        print(f"Review: '{review_text[:100]}...'")
        print(f"True sentiment: {true_sentiment}")
        print(f"Predicted sentiment: {pred_sentiment} (Score: {y_pred_proba[idx][0]:.4f})")
        print()

print("Analysis completed.")
```

```
Downloading NLTK stopwords...
Loading dataset...
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz
 8036352/17464789 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
17464789/17464789 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb_word_index.json
1641221/1641221 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step

Sample reviews from the dataset:
Review 1 (Sentiment: Positive):
this film was just brilliant casting location scenery story direction everyone's really suited the part they played and you cou
Review 2 (Sentiment: Negative):
big hair big boobs bad music and a giant safety pin these are the words to best describe this terrible movie i love cheesy horr
Review 3 (Sentiment: Negative):
this has to be one of the worst films of the 1990s when my friends i were watching this film being the target audience it was a
Review 4 (Sentiment: Positive):
the at storytelling the traditional sort many years after the event i can still see in my eye an elderly lady my friend's mother
Review 5 (Sentiment: Negative):
worst mistake of my life br br i picked this movie up at target for 5 because i figured hey it's sandler i can get some cheap la

Training dataset size: 25000 reviews
Testing dataset size: 25000 reviews

Class distribution in training data:
Negative: 12500 (50.0%)
Positive: 12500 (50.0%)

Preprocessing data...
Training data shape after padding: (25000, 200)
Testing data shape after padding: (25000, 200)

Building LSTM model...
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecate
  warnings.warn(
Model: "sequential"
```

Layer (type)	Output Shape	Param #
embedding (Embedding)	?	0 (unbuilt)
bidirectional (Bidirectional)	?	0 (unbuilt)
dropout (Dropout)	?	0
bidirectional_1 (Bidirectional)	?	0 (unbuilt)
dropout_1 (Dropout)	?	0
dense (Dense)	?	0 (unbuilt)

```
 Total params: 0 (0.00 B)
 Trainable params: 0 (0.00 B)
 Non-trainable params: 0 (0.00 B)

Training the model...
Epoch 1/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 204s 628ms/step - accuracy: 0.6815 - loss: 0.5695 - val_accuracy: 0.8188 - val_loss: 0.4001
Epoch 2/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 190s 608ms/step - accuracy: 0.8774 - loss: 0.3004 - val_accuracy: 0.8612 - val_loss: 0.3791
Epoch 3/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 203s 613ms/step - accuracy: 0.9014 - loss: 0.2612 - val_accuracy: 0.8490 - val_loss: 0.4233
Epoch 4/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 214s 651ms/step - accuracy: 0.9237 - loss: 0.2012 - val_accuracy: 0.7992 - val_loss: 0.4924
Epoch 5/10
313/313 ━━━━━━━━━━━━━━━━━━━━ 251s 617ms/step - accuracy: 0.9388 - loss: 0.1678 - val_accuracy: 0.7970 - val_loss: 0.5030

Evaluating the model...
782/782 ━━━━━━━━━━━━━━━━━━━━ 57s 73ms/step - accuracy: 0.8429 - loss: 0.4200
Test Loss: 0.4190
Test Accuracy: 0.8426
782/782 ━━━━━━━━━━━━━━━━━━━━ 57s 72ms/step

Classification Report:
              precision    recall  f1-score   support
```