Objective: Sentiment Classification Reviews categorized into positive, negative, or neutral sentiments, evaluated using accuracy

Previously, we have compared three models for sentiment classidication on a subset of the raw dataset: Logistic Regression, LSTM models and DistilBERT. Considering best performce for malti-class analysis and time consuming, we chose to apply the LSTM model on the whole dataset and it can be found in https://github.com/sondhia/amazonfinefood/blob/main/sentiment/scr/LSTM%20on%20whole%20dataset.ip

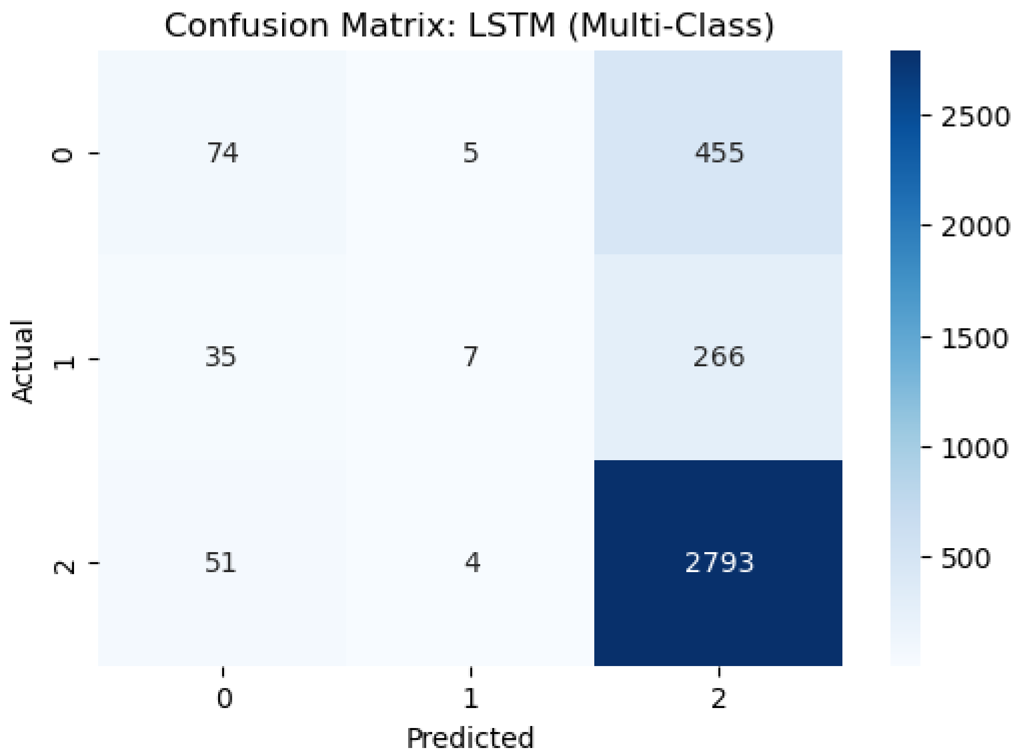
Evaluate the LSTM Model

Here we use f1-score, accuracy, and confusion matrix to evaluate the LSTM model

In [29]:

```
# Predict probabilities on the test set and convert to predicted class indices
y_pred_probs = lstm_model.predict(X_test_pad)
y_pred = np.argmax(y_pred_probs, axis=1)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
weighted_f1 = f1_score(y_test, y_pred, average='weighted')
print("\n--- LSTM Model Evaluation (Multi-Class) ---")
print("Accuracy: {:.4f}".format(accuracy))
print("Weighted F1 Score: {:.4f}".format(weighted_f1))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix: LSTM (Multi-Class)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

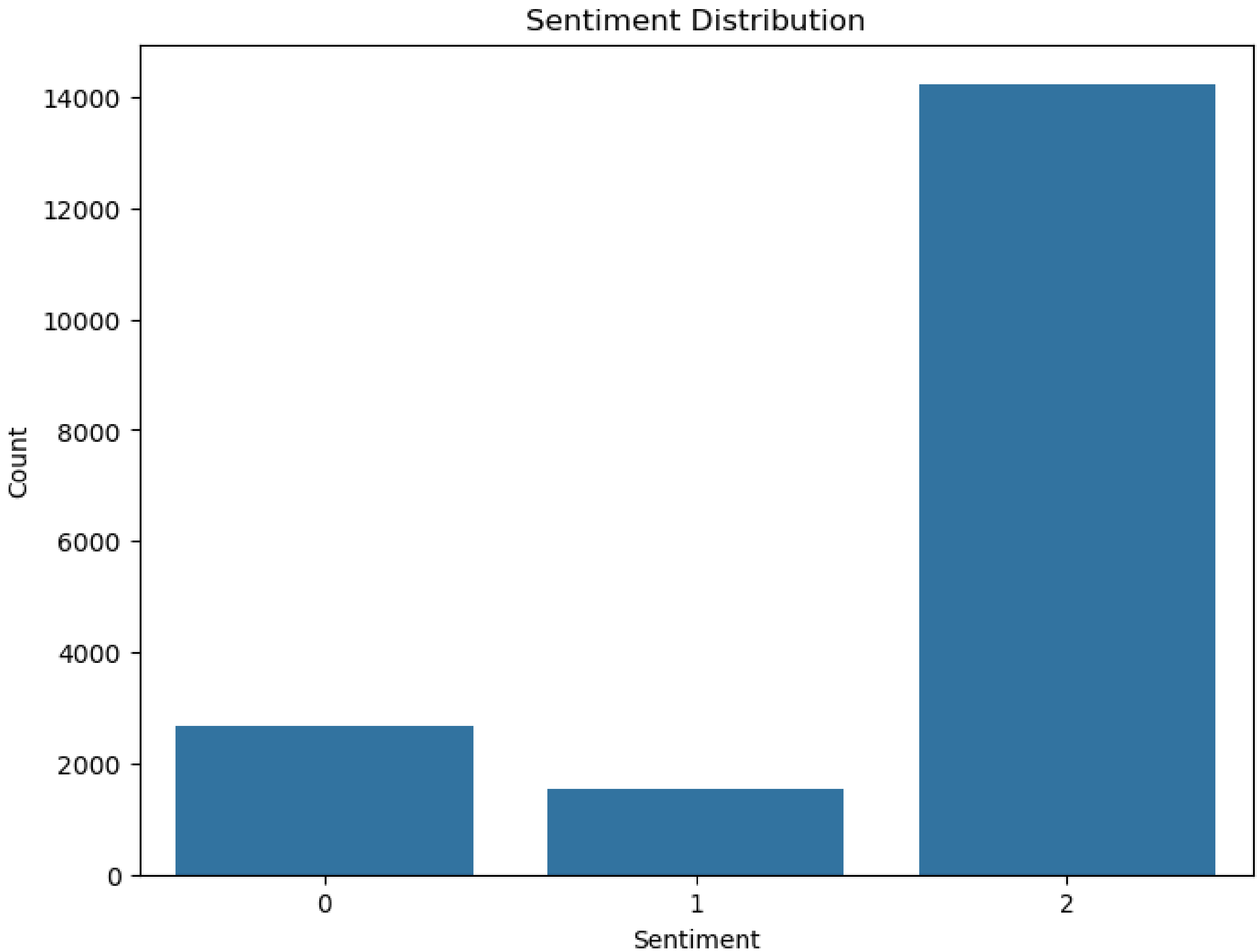
```
116/116 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step

--- LSTM Model Evaluation (Multi-Class) ---
Accuracy: 0.7789
Weighted F1 Score: 0.7121

Classification Report:
               precision    recall  f1-score   support

           0       0.46      0.14      0.21       534
           1       0.44      0.02      0.04       308
           2       0.79      0.98      0.88      2848

    accuracy                           0.78      3690
   macro avg       0.56      0.38      0.38      3690
weighted avg       0.72      0.78      0.71      3690
```



From the classification report and confusion matrix, we can see the f1-score of 0 and 1 is significant low which could due to the imbalanced calss in the data set. Next, we checked the sentiment distrbution and found most of the sentiment are 2 (Positive)

In [31]:

```
# Plot the sentiment distribution
plt.figure(figsize=(8,6))
sns.countplot(x='Sentiment', data=df)
plt.title('Sentiment Distribution')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.show()
```



To address the imbalance issue, we compute each class weights among the training set and it will be used in the uopdated model training

In [33]:
```
# Compute class weights based on the original training set
classes = np.unique(y_train)
class_weights = compute_class_weight(class_weight='balanced', classes=classes, y=y_train
class_weight_dict = dict(zip(classes, class_weights))
print("\nComputed class weights:")
print(class_weight_dict)
```

```
Computed class weights:
{0: 2.301669006395258, 1: 3.989186266558529, 2: 0.431992505416008}
```

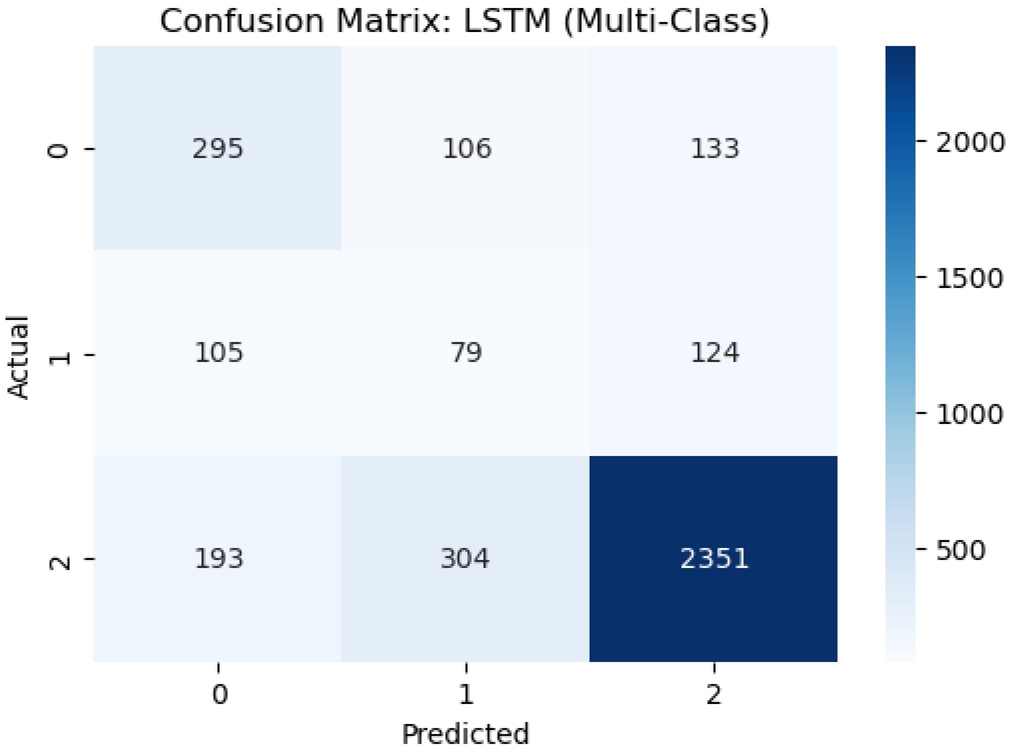
In [35]:
```
# Train the model
history = lstm_model.fit(
    X_train_pad, np.array(y_train),
    epochs=5,
    batch_size=32,
    validation_split=0.1,
    class_weight=class_weight_dict,
    verbose=2
)
```

```
Epoch 1/5
415/415 - 9s - 22ms/step - accuracy: 0.6940 - loss: 0.9407 - val_accuracy: 0.4031 - val_l
oss: 1.0270
Epoch 2/5
415/415 - 9s - 21ms/step - accuracy: 0.7387 - loss: 0.8895 - val_accuracy: 0.7446 - val_l
oss: 0.8504
Epoch 3/5
415/415 - 9s - 21ms/step - accuracy: 0.6903 - loss: 0.8722 - val_accuracy: 0.7940 - val_l
oss: 0.8893
Epoch 4/5
415/415 - 9s - 21ms/step - accuracy: 0.7733 - loss: 0.7578 - val_accuracy: 0.7737 - val_l
oss: 0.7437
Epoch 5/5
415/415 - 9s - 21ms/step - accuracy: 0.8275 - loss: 0.7231 - val_accuracy: 0.7561 - val_l
oss: 0.8604
```

In [37]:

```
#Evaluate the LSTM Model
# Predict probabilities on the test set and convert to predicted class indices
y_pred_probs = lstm_model.predict(X_test_pad)
y_pred = np.argmax(y_pred_probs, axis=1)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
weighted_f1 = f1_score(y_test, y_pred, average='weighted')
print("\n--- LSTM Model Evaluation (Multi-Class) ---")
print("Accuracy: {:.4f}".format(accuracy))
print("Weighted F1 Score: {:.4f}".format(weighted_f1))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix: LSTM (Multi-Class)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
116/116 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step

--- LSTM Model Evaluation (Multi-Class) ---
Accuracy: 0.7385
Weighted F1 Score: 0.7575

Classification Report:
               precision    recall  f1-score   support

           0       0.50      0.55      0.52       534
           1       0.16      0.26      0.20       308
           2       0.90      0.83      0.86      2848

    accuracy                           0.74      3690
   macro avg       0.52      0.54      0.53      3690
weighted avg       0.78      0.74      0.76      3690
```



The LSTM model is trained by class_weight and the evaluation report show better f1-score in each class.

Model improvment

Next we try to use different hyperparameters—the number of LSTM units, dropout rates, and additional layer to improve the model accuracy. A loop is used to train models with different settings and compare the accuracy.

In [53]:
```
# Hyperparameter grid
lstm_units_list = [32, 64, 128]
dropout_rates = [0.2, 0.3, 0.5]
stacked_options = [False, True]

best_val_acc = 0
best_config = None

for units in lstm_units_list:
    for dropout in dropout_rates:
        for stacked in stacked_options:
            print(f"Training model with LSTM units: {units}, Dropout: {dropout}, Stacked
            model = Sequential()
            model.add(Embedding(input_dim=10000, output_dim=64))
            # First LSTM layer with return_sequences if stacking is desired
            model.add(LSTM(units, return_sequences=stacked))
            model.add(Dropout(dropout))
            # Optional second LSTM layer
            if stacked:
                model.add(LSTM(units))
                model.add(Dropout(dropout))
            model.add(Dense(len(np.unique(y_train)), activation='softmax'))
            model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metr

            history = model.fit(X_train_pad, np.array(y_train),
                                epochs=5,
```

```
                                           batch_size=32,
                                           validation_split=0.1,
                                           class_weight=class_weight_dict,
                                           verbose=0)
                val_acc = history.history['val_accuracy'][-1]
                print(f"Validation Accuracy: {val_acc:.4f}")
                if val_acc > best_val_acc:
                    best_val_acc = val_acc
                    best_config = (units, dropout, stacked)

print("\nBest configuration:")
print(f"LSTM Units: {best_config[0]}, Dropout: {best_config[1]}, Stacked: {best_config[2
```

```
Training model with LSTM units: 32, Dropout: 0.2, Stacked: False
Validation Accuracy: 0.6829
Training model with LSTM units: 32, Dropout: 0.2, Stacked: True
Validation Accuracy: 0.1992
Training model with LSTM units: 32, Dropout: 0.3, Stacked: False
Validation Accuracy: 0.7622
Training model with LSTM units: 32, Dropout: 0.3, Stacked: True
Validation Accuracy: 0.6795
Training model with LSTM units: 32, Dropout: 0.5, Stacked: False
Validation Accuracy: 0.7215
Training model with LSTM units: 32, Dropout: 0.5, Stacked: True
Validation Accuracy: 0.7256
Training model with LSTM units: 64, Dropout: 0.2, Stacked: False
Validation Accuracy: 0.6755
Training model with LSTM units: 64, Dropout: 0.2, Stacked: True
Validation Accuracy: 0.7344
Training model with LSTM units: 64, Dropout: 0.3, Stacked: False
Validation Accuracy: 0.5874
Training model with LSTM units: 64, Dropout: 0.3, Stacked: True
Validation Accuracy: 0.7466
Training model with LSTM units: 64, Dropout: 0.5, Stacked: False
Validation Accuracy: 0.3035
Training model with LSTM units: 64, Dropout: 0.5, Stacked: True
Validation Accuracy: 0.6728
Training model with LSTM units: 128, Dropout: 0.2, Stacked: False
Validation Accuracy: 0.7432
Training model with LSTM units: 128, Dropout: 0.2, Stacked: True
Validation Accuracy: 0.6592
Training model with LSTM units: 128, Dropout: 0.3, Stacked: False
Validation Accuracy: 0.7249
Training model with LSTM units: 128, Dropout: 0.3, Stacked: True
Validation Accuracy: 0.7812
Training model with LSTM units: 128, Dropout: 0.5, Stacked: False
Validation Accuracy: 0.7358
Training model with LSTM units: 128, Dropout: 0.5, Stacked: True
Validation Accuracy: 0.6504

Best configuration:
LSTM Units: 128, Dropout: 0.3, Stacked: True, Val Accuracy: 0.7812
```

The Best configuration is applied to train the model (LSTM Units: 128, Dropout: 0.3, Stacked: True, Val Accuracy: 0.7812)

In [57]:

```
# Build a stacked LSTM model with best configuration
lstm_best_model = Sequential([
    Embedding(input_dim=10000, output_dim=64),
    LSTM(128, return_sequences=True),
    Dropout(0.3),
```

```
    LSTM(128),
    Dropout(0.3),
    Dense(num_classes, activation='softmax')
])

lstm_best_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metric
lstm_best_model.summary()
```

Model: "sequential_22"

Layer (type)	Output Shape	Param #
embedding_22 (Embedding)	?	0 (unbuilt)
lstm_34 (LSTM)	?	0 (unbuilt)
dropout_34 (Dropout)	?	0
lstm_35 (LSTM)	?	0 (unbuilt)
dropout_35 (Dropout)	?	0
dense_22 (Dense)	?	0 (unbuilt)

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

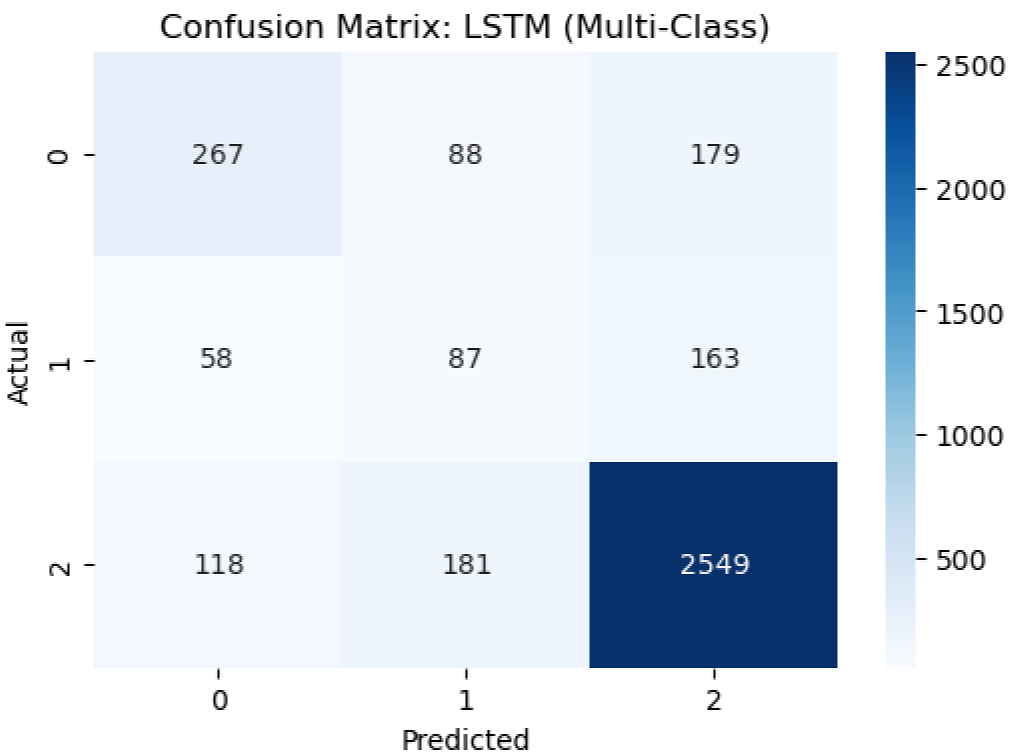
In [63]:
```
history3 = lstm_best_model.fit(
    X_train_pad, np.array(y_train),
    epochs=5,
    batch_size=32,
    validation_split=0.1,
    class_weight=class_weight_dict,
    verbose=2
)
```

```
Epoch 1/5
415/415 - 48s - 117ms/step - accuracy: 0.9855 - loss: 0.0558 - val_accuracy: 0.7514 - val
_loss: 1.1678
Epoch 2/5
415/415 - 49s - 118ms/step - accuracy: 0.9831 - loss: 0.0601 - val_accuracy: 0.8015 - val
_loss: 1.0470
Epoch 3/5
415/415 - 51s - 123ms/step - accuracy: 0.9812 - loss: 0.0738 - val_accuracy: 0.7656 - val
_loss: 1.0878
Epoch 4/5
415/415 - 53s - 127ms/step - accuracy: 0.9880 - loss: 0.0440 - val_accuracy: 0.8062 - val
_loss: 0.9673
Epoch 5/5
415/415 - 49s - 118ms/step - accuracy: 0.9888 - loss: 0.0418 - val_accuracy: 0.8062 - val
_loss: 1.0789
```

In [65]:
```
#Evaluate the stacked LSTM Model
# Predict probabilities on the test set and convert to predicted class indices
y_pred_probs = lstm_best_model.predict(X_test_pad)
y_pred = np.argmax(y_pred_probs, axis=1)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
weighted_f1 = f1_score(y_test, y_pred, average='weighted')
print("\n--- LSTM Model Evaluation (Multi-Class) ---")
```

```
print("Accuracy: {:.4f}".format(accuracy))
print("Weighted F1 Score: {:.4f}".format(weighted_f1))
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Plot confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix: LSTM (Multi-Class)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
116/116 ━━━━━━━━━━━━━━━━━━━━ 5s 39ms/step

--- LSTM Model Evaluation (Multi-Class) ---
Accuracy: 0.7867
Weighted F1 Score: 0.7866

Classification Report:
               precision    recall  f1-score   support

           0       0.60      0.50      0.55       534
           1       0.24      0.28      0.26       308
           2       0.88      0.90      0.89      2848

    accuracy                           0.79      3690
   macro avg       0.58      0.56      0.57      3690
weighted avg       0.79      0.79      0.79      3690
```



Conclusion

The model was trained on Amazon review data, which was preprocessed by lowercasing, removing HTML tags and non-alphabetic characters, and eliminating extra spaces. Due to class imbalance, we applied class weighting during training. The goal is to improve the model's effectiveness in classifying sentimen.

Model Performance:

- The LSTM model effectively captures sequential patterns in text, leading to a robust baseline performance.
- Preprocessing steps such as lowercasing and cleaning contribute to the model's ability to focus on the core content of the reviews.
- Class weighting has helped mitigate the imbalance issue, allowing the model to learn features from underrepresented classes.

Limitation and Improvement:

- Despite class weighting, the inherent imbalance still causes the minority class (1 Netural) instances to be misclassified.
- The current model is relatively simple. There is potential for improvement by stacking more LSTM layers or using bidirectional LSTMs.

```
In [ ]:
```