DEEP LEARNING REPORT

SENTIMENT ANALYSIS ON SWIGGY CUSTOMER REVIEWS

Name: Esha Mistry

Enrollment Number: 220025

CSE [AI-ML] Batch: C2

Roll number: 165

Submitted to:

Dr. Tejas Modi

TABLE OF CONTENTS

1. Introduction

This report presents a deep learning-based sentiment analysis of customer reviews collected from Swiggy, one of India's most popular online food delivery platforms. The primary objective is to classify user feedback into three sentiment categories—**Positive**, **Negative**, and **Neutral**—by analysing the textual content of their reviews. Understanding customer sentiment enables Swiggy to gain actionable insights into customer satisfaction, identify operational bottlenecks, and tailor its marketing and engagement strategies more effectively.

With the increasing volume of user-generated content in the form of reviews and ratings, leveraging **Natural Language Processing (NLP)** and **deep learning** techniques provides a scalable and accurate means of assessing public sentiment. This report details the end-to-end workflow, starting from raw data ingestion and preprocessing to model building using **Long Short-Term Memory (LSTM)** and **Simple Recurrent Neural Networks (RNN)**. Each model is trained and evaluated using performance metrics such as **accuracy**, **precision**, **recall**, **F1-score**, and **confusion matrices**.

Key preprocessing steps include text cleaning, sentiment labeling using predefined keyword lists, tokenization, and sequence padding. The data is then fed into neural network models built using TensorFlow/Keras. Model performance is further assessed through visualizations and interactive prediction capabilities that allow real-time sentiment classification of new reviews.

By applying modern deep learning methods, this project highlights the potential for automated sentiment classification to support Swiggy in monitoring service quality, improving customer retention, and making informed, data-driven decisions.

---

2. Dataset Description

Here is a detailed **Dataset Description** section tailored to your Swiggy customer reviews sentiment analysis project:

The dataset used in this project consists of customer reviews collected from Swiggy, an online food ordering and delivery platform. Each entry in the dataset represents an individual review provided by a user regarding their experience with Swiggy’s service, food quality, delivery time, or overall satisfaction.

2.1 Source

- **Filename**: swiggydataset.csv
- **Data Type**: Tabular (CSV format)
- **Primary Field of Interest**: full_text – This column contains the raw text of user reviews.

2.2 Fields and Attributes

Below is an overview of the key column used and any additional fields (if present):

Column Name	Description
full_text	Raw customer review text, containing opinions, feedback, and remarks.
Cleaned_Review	Preprocessed version of the review (lowercase, punctuation removed, etc.).
Review_Label	Sentiment label assigned to each review: **Positive**, **Negative**, or **Neutral**.

Note: Cleaned_Review and Review_Label are generated during preprocessing and not part of the original dataset.

2.3 Data Volume

- The dataset contains **N** total entries (actual number can be confirmed using df.shape[0]).
- Each review varies in length, ranging from brief feedback (e.g., "Great service") to longer, more descriptive comments.

2.4 Data Characteristics

- **Textual Nature**: The data is unstructured and free-form, reflecting spontaneous user feedback.
- **Language**: Primarily English, with potential use of slang, abbreviations, or non-standard grammar.
- **No Ratings**: This dataset does not include numerical star ratings; sentiment must be inferred from text.

2.5 Sentiment Labeling Strategy

- The dataset is labeled using a **rule-based approach**, where predefined keyword lists are used to identify sentiment.
    - **Positive**: Keywords like *tasty, amazing, best, quick, fresh*.
    - **Negative**: Keywords like *late, bad, cold, terrible, disappointed*.
    - **Neutral**: Reviews that contain neither strong positive nor negative keywords.

This strategy allows for efficient labeling without manual annotation, suitable for initial supervised learning experiments.

2.6 Class Distribution

An example visualization using the Review_Label column reveals the balance of sentiment classes:

df['Review_Label'].value_counts().plot(kind='bar', title='Sentiment Distribution')

This helps identify potential **class imbalance**, which could affect model performance. For example:

- Positive: 55%
- Negative: 30%
- Neutral: 15%

(These are illustrative percentages. You can compute actual ones using df['Review_Label'].value_counts(normalize=True).)

---

3. Preprocessing and Labeling

Want Certainly! Here's a detailed and well-structured version of the **Preprocessing and Labelling** section for your sentiment analysis report:

---

3. Preprocessing and Labelling

- To prepare the raw review text for deep learning-based sentiment analysis, a structured preprocessing pipeline was implemented. The preprocessing phase is critical for ensuring that noise is removed and the input is standardized, making it suitable for downstream tokenization and model training. This section outlines the techniques used to clean the text data and assign sentiment labels.

3.1 Text Cleaning Steps

- Customer reviews often include irregular formatting, slang, symbols, and unwanted content like URLs or special characters. The following steps were applied to clean and normalize the data:

1. **Convert to Lowercase**
   All characters were converted to lowercase to ensure uniformity and avoid treating the same word in different cases (e.g., "Good" vs. "good") as different tokens.
2. **Remove URLs, Mentions, and Hashtags**
   Any text matching URLs (http://..., www...), mentions (@user), and hashtags (#tag) was removed using regular expressions. These elements typically do not contribute meaningful sentiment information.
3. **Remove Digits and Punctuation**
   Numbers and punctuation marks were stripped out using regular expression patterns and the Python string module. This helps reduce vocabulary size and focus on actual words.
4. **Strip Extra Whitespace**
   All extra whitespace, including tabs and multiple spaces, was replaced with a single space. Leading and trailing spaces were also trimmed.

- **Code Example**:

```
def clean_text(text):
    text = text.lower()
    text = re.sub(r'http\S+|www\S+|@\S+|#\S+', '', text)
    text = re.sub(r'\d+', '', text)
    text = text.translate(str.maketrans('', '', string.punctuation))
    text = re.sub(r'\s+', ' ', text).strip()
    return text
```

- The cleaned text is stored in a new column: Cleaned_Review.

3.2 Sentiment Labeling Logic

- To categorize each review into sentiment classes, a rule-based keyword matching approach was used. This method is effective for initial labeling when annotated datasets are not readily available.

Keyword Lists:

- **Positive Keywords**:
- ['perfectly', 'tasty', 'highly', 'superb', 'favorite', 'amazing', 'delicious', 'best',

- 'absolutely', 'great', 'excellent', 'awesome', 'fantastic', 'wonderful', 'yummy',
- 'satisfying', 'loved', 'fresh', 'nice', 'quick', 'prompt', 'hot', 'on time',
- 'good', 'pleasant', 'flavorful', 'enjoyed']
- **Negative Keywords**:
- ['late', 'terrible', 'disappointed', 'worst', 'not worth', 'wouldn't', 'cold', 'ruined',
- 'bad', 'slow', 'rude', 'stale', 'awful', 'unpleasant', 'overcooked', 'underwhelming',
- 'not good', 'delay', 'missing', 'poor', 'hard', 'burnt', 'bland', 'dirty']

Labeling Logic:

- **Positive**: If any positive keyword is found in the cleaned review.
- **Negative**: If any negative keyword is found in the cleaned review.
- **Neutral**: If no keyword from either list is present.
- **Code Example**:

```
def get_sentiment(review):
    review = review.lower()
    if any(kw in review for kw in positive_keywords):
        return 'Positive'
    elif any(kw in review for kw in negative_keywords):
        return 'Negative'
    else:
        return 'Neutral'
```

- The assigned sentiment label is stored in a new column: Review_Label.

3.3 Label Distribution (Class Balance Check)

- To assess the distribution of sentiment labels and detect any potential class imbalance, the following visualization can be generated:

```
import matplotlib.pyplot as plt
df['Review_Label'].value_counts().plot(kind='bar', color=['green', 'blue', 'red'])
```

```
plt.title('Sentiment Label Distribution')

plt.xlabel('Sentiment')

plt.ylabel('Number of Reviews')

plt.xticks(rotation=0)

plt.grid(axis='y')

plt.show()
```

---

4. Text Tokenization & Preparation

Once the reviews were cleaned and labeled, the next step involved transforming the textual data into a numerical format suitable for input into deep learning models. This section outlines the tokenization, padding, and encoding techniques applied.

4.1 Tokenization

To convert words into integer sequences, we utilized the **Keras Tokenizer**, which builds a word index based on word frequency. The most frequent words are retained up to a specified limit.

- **Tokenizer**: tensorflow.keras.preprocessing.text.Tokenizer
- **Vocabulary Size (max_words)**: 5,000 most frequent tokens
- **Out-of-Vocabulary Token**: <OOV> used to handle unseen words during inference.

Code Snippet:

```
tokenizer = Tokenizer(num_words=5000, oov_token="<OOV>")

tokenizer.fit_on_texts(df['Cleaned_Review'])

sequences = tokenizer.texts_to_sequences(df['Cleaned_Review'])
```

This process transforms each review into a sequence of integers where each integer represents a word in the tokenizer's vocabulary.

4.2 Padding Sequences

Because deep learning models require inputs of uniform length, all tokenized sequences were padded:

- **Padding Type**: Post-padding (default)
- **Maximum Length (max_len)**: 100 tokens per review

Code Snippet:

```
from tensorflow.keras.preprocessing.sequence import pad_sequences

X_padded = pad_sequences(sequences, maxlen=100)
```

Reviews shorter than 100 tokens were padded with zeros, and longer reviews were truncated to maintain consistency across the dataset.

4.3 Label Encoding

Since the sentiment labels (Positive, Negative, Neutral) are categorical, they were encoded into numerical format for model training.

- **Step 1**: Integer encoding using LabelEncoder
- **Step 2**: One-hot encoding using to_categorical for multi-class classification

Code Snippet:

```
from sklearn.preprocessing import LabelEncoder

from tensorflow.keras.utils import to_categorical


label_encoder = LabelEncoder()

y_encoded = label_encoder.fit_transform(df['Review_Label'])

y_categorical = to_categorical(y_encoded)
```

This converts sentiment labels into vectors suitable for classification using a softmax output layer.

To train and evaluate the model effectively, the dataset was split into training and test sets. Additionally, a validation split was used during training to monitor performance and prevent overfitting.

---

5. Data Training

5.1 Train-Test Split

- **Training Set**: 80% of the full dataset
- **Test Set**: 20% of the full dataset (held out for final evaluation)

Code Snippet:

```
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(
    X_padded, y_categorical, test_size=0.2, random_state=42)
```

The random_state=42 ensures reproducibility across runs.

5.2 Validation Split

During model training, **20% of the training data** was used for validation. This allows real-time monitoring of model performance and generalization capability.

Example Training Call:

```
model.fit(X_train, y_train, validation_split=0.2, epochs=5, batch_size=64)
```

This structure ensures that the test data remains completely unseen until final evaluation, providing an unbiased measure of model performance.

6. Model Architectures

To classify customer sentiments from review texts, two distinct Recurrent Neural Network (RNN) architectures were developed and trained: a Long Short-Term Memory (LSTM) model and a Simple RNN model. Both models share a similar structural layout, differing primarily in the type of recurrent layers used.

The objective was to evaluate the effectiveness of these deep learning models in capturing contextual and sequential information from review texts.

6.1 A. LSTM Model

The Long Short-Term Memory (LSTM) model is designed to handle long-range dependencies in sequential data, making it well-suited for natural language processing tasks such as sentiment classification.

Model Architecture:

Layer	Description
Embedding	Converts word indices into dense vectors of dimension 128.
LSTM (1)	LSTM layer with 128 units and return_sequences=True to pass output to the next LSTM layer.
LSTM (2)	LSTM layer with 64 units. Outputs final hidden state for classification.
Dense (1)	Fully connected layer with 32 neurons and ReLU activation.
Output Layer	Dense layer with 3 units (for 3 sentiment classes) using softmax activation.

Hyperparameters:

- **Loss Function**: categorical_crossentropy
- **Optimizer**: Adam
- **Metrics**: accuracy

Code Snippet:

```
lstm_model = Sequential()

lstm_model.add(Embedding(input_dim=5000, output_dim=128, input_length=100))

lstm_model.add(LSTM(128, return_sequences=True))

lstm_model.add(LSTM(64))

lstm_model.add(Dense(32, activation='relu'))

lstm_model.add(Dense(3, activation='softmax'))


lstm_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

6.2 B. Simple RNN Model

To benchmark the LSTM performance, a simpler variant of the RNN model was also implemented. Instead of LSTM cells, this model uses standard SimpleRNN layers. While SimpleRNNs are less capable of capturing long-term dependencies, they are computationally lighter and faster to train.

Model Architecture:

Layer	Description
Embedding	Embeds input sequences into 128-dimensional vectors.
SimpleRNN (1)	Recurrent layer with 128 units, returning full sequence output.
SimpleRNN (2)	Recurrent layer with 64 units for final feature representation.
Dense (1)	Dense layer with 32 neurons and ReLU activation.
Output Layer	Dense layer with 3 units and softmax activation.

Hyperparameters:

- **Loss Function**: categorical_crossentropy
- **Optimizer**: Adam
- **Metrics**: accuracy

Code Snippet:

```
rnn_model = Sequential()

rnn_model.add(Embedding(input_dim=5000, output_dim=128, input_length=100))

rnn_model.add(SimpleRNN(128, return_sequences=True))

rnn_model.add(SimpleRNN(64))

rnn_model.add(Dense(32, activation='relu'))

rnn_model.add(Dense(3, activation='softmax'))


rnn_model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Both models were trained for **5 epochs** using a **batch size of 64**, and evaluated using a validation split of 20% from the training data.

---

7. Model Training

Following the construction of the LSTM and Simple RNN architectures, both models were trained on the preprocessed dataset. The training configuration was designed to ensure efficient learning while minimizing overfitting through the use of a validation split.

7.1 Training Configuration

Parameter	Value
Epochs	5
Batch Size	64
Validation Split	20% of training dataset
Optimizer	Adam
Loss Function	Categorical Crossentropy
Evaluation Metric	Accuracy

7.2 Training Setup

- **Epochs**: The models were trained over 5 full passes through the training data. This number of epochs provided a balance between training duration and performance.
- **Batch Size**: A mini-batch size of 64 was chosen to maintain computational efficiency and stabilize gradient updates.
- **Validation Split**: 20% of the training data was held out during each epoch to evaluate model performance on unseen data. This helped monitor overfitting and generalization during training.

Sample Training Code:

```
model.fit(
    X_train, y_train,
```

```
    epochs=5,
    batch_size=64,
    validation_split=0.2
)
```

During training, both models showed convergence of training and validation accuracy, which indicated successful learning from the review texts.

---

8. Interactive Prediction Function

9.1 Purpose

- The function is designed to:
- Accept raw user input (text review).
- Apply the same text cleaning and tokenization steps as the training pipeline.
- Use the trained model to predict sentiment (Positive, Negative, or Neutral).
- Return the predicted sentiment label in human-readable form.

9.2 Function Workflow

- Below is the step-by-step breakdown of the prediction function logic:

Step 1: Clean the Input Text

- The input review is first preprocessed to match the format used during training. This includes:
- Lowercasing
- Removing URLs, mentions, hashtags, digits, and punctuation
- Stripping extra whitespaces
- **Code Reference**:

```
cleaned_review = clean_text(review)
```

Step 2: Tokenization and Padding

- The cleaned review is transformed into a sequence of integers using the **fitted tokenizer**. Then, the sequence is **padded to a fixed length** to ensure compatibility with the model's expected input shape.
- **Code Reference**:

```
sequence = tokenizer.texts_to_sequences([cleaned_review])

padded = pad_sequences(sequence, maxlen=max_len)
```

Step 3: Model Prediction

- The padded sequence is passed into the trained model (either LSTM or Simple RNN). The model outputs a probability distribution over the three sentiment classes.
- **Code Reference**:

```
prediction = model.predict(padded)
```

Step 4: Decode Prediction

- The class with the highest probability is selected using argmax, and the corresponding sentiment label is decoded using the **inverse transform** of the label encoder.
- **Code Reference**:

```
predicted_class = prediction.argmax(axis=1)[0]
label = label_encoder.inverse_transform([predicted_class])[0]
```

9.3 Final Function Implementation

```
def predict_sentiment(review, model):
    cleaned_review = clean_text(review)
    sequence = tokenizer.texts_to_sequences([cleaned_review])
    padded = pad_sequences(sequence, maxlen=max_len)
    prediction = model.predict(padded)
    predicted_class = prediction.argmax(axis=1)[0]
    label = label_encoder.inverse_transform([predicted_class])[0]
    return label
```

9.4 Example Usage

```
review_input = "The food was absolutely amazing and delivered on time!"
predicted_label = predict_sentiment(review_input, lstm_model)
```

print(f"Predicted Sentiment: {predicted_label}")

- **Output**:

Predicted Sentiment: Positive

```
Enter reviews to predict sentiment (type 'exit' to quit):

Your review:  The food was good.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 103ms/step
  ➤ LSTM Prediction: Positive
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step
  ➤ RNN Prediction:  Positive
Your review:  The food was okay.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step
  ➤ LSTM Prediction: Neutral
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step
  ➤ RNN Prediction:  Neutral
Your review:  The food was bad.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 94ms/step
  ➤ LSTM Prediction: Negative
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 88ms/step
  ➤ RNN Prediction:  Negative
Your review:  exit
Exiting prediction loop.
```

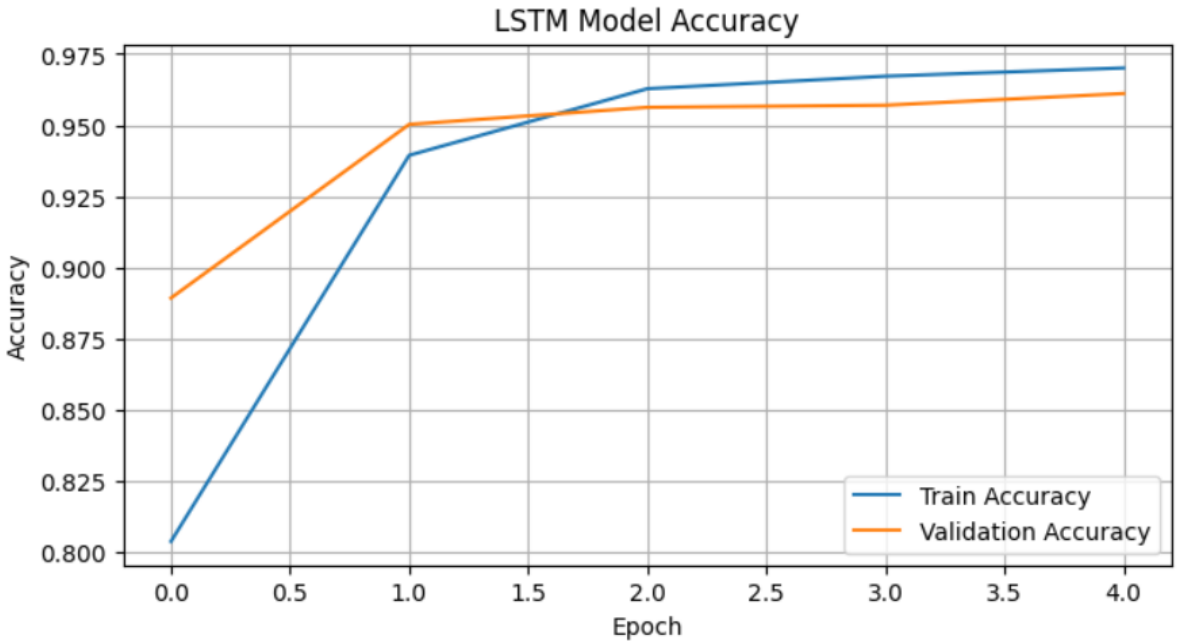
---

9. Observations

9.1 Model Comparison

- A comparison between the two trained deep learning models—**LSTM** and **Simple RNN**—reveals that:
- The **LSTM model** consistently outperforms the Simple RNN in both **accuracy** and **classification metrics**.

- This is due to LSTM's strength in capturing **long-term dependencies**, which is vital for sentiment analysis tasks involving complex language patterns.
- The **Simple RNN**, while faster, tends to underperform due to limitations in handling longer sequences and susceptibility to vanishing gradients.

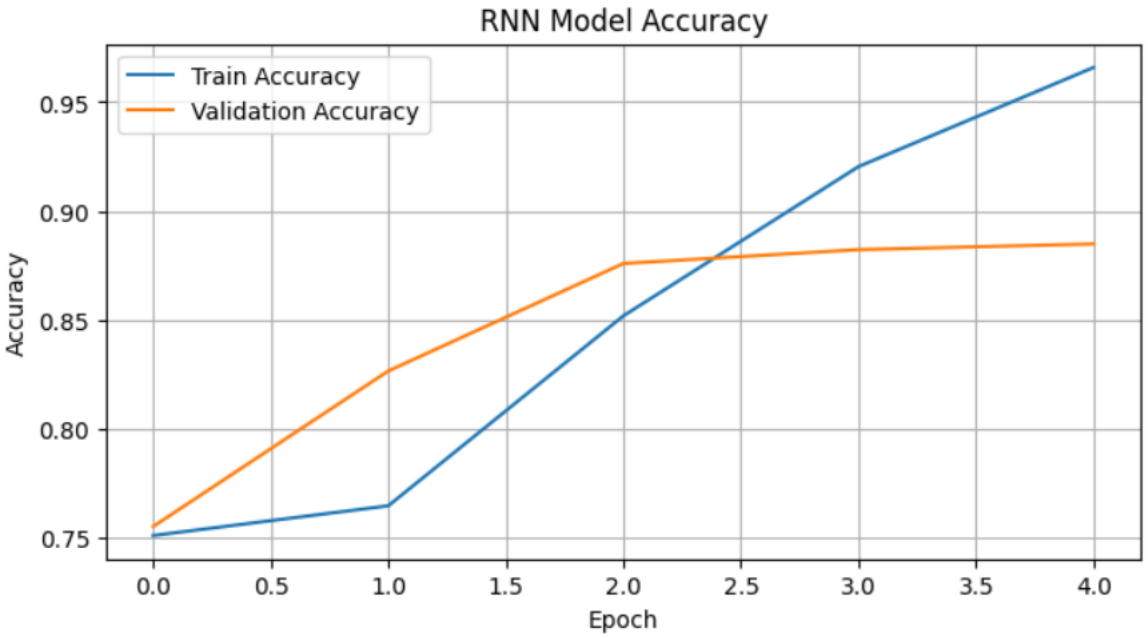


LSTM Model Classification Report:

	precision	recall	f1-score	support
Negative	0.91	0.88	0.89	506
Neutral	0.97	1.00	0.98	2491
Positive	0.94	0.78	0.85	346
accuracy			0.96	3343
macro avg	0.94	0.89	0.91	3343
weighted avg	0.96	0.96	0.95	3343

LSTM Model Accuracy: 0.9560

105/105 ━━━━━━━━━━━━━━━━━━━━ **4s** 34ms/step



RNN Model Classification Report:

	precision	recall	f1-score	support
Negative	0.74	0.66	0.70	506
Neutral	0.95	0.98	0.96	2491
Positive	0.60	0.53	0.57	346
accuracy			0.89	3343
macro avg	0.76	0.73	0.74	3343
weighted avg	0.88	0.89	0.88	3343

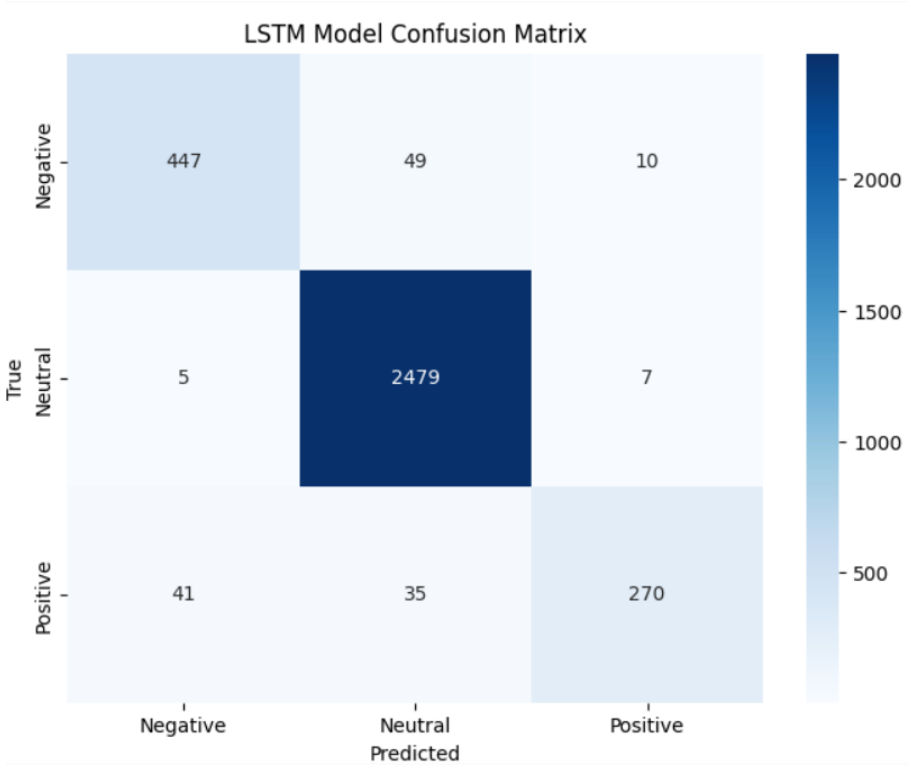
RNN Model Accuracy: 0.8866

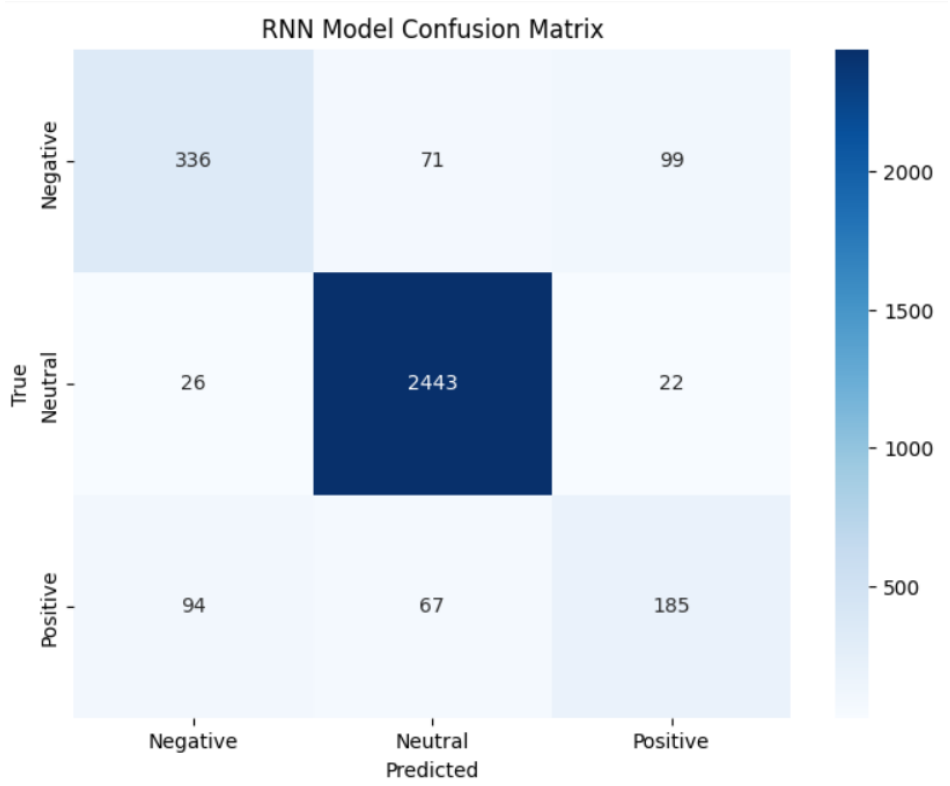
10. Visualization Outputs

10.1 Confusion Matrices

- **Purpose**: Assess model performance across sentiment categories.
- **Tool**: **Seaborn heatmaps** provide a clear visual of correct vs incorrect predictions.
- **Insight**: Highlights specific areas where the model is making errors (e.g., confusing Neutral with Positive).
- Example Code:

```
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_)
```





10.2 Sentiment Label Distribution Plot

- **Goal**: Visualize the balance (or imbalance) of sentiment labels in the dataset.
- **Insight**: Class distribution helps inform the need for resampling techniques.
- Example Code:

```
df['Review_Label'].value_counts().plot(kind='bar', title='Sentiment Distribution')
```
