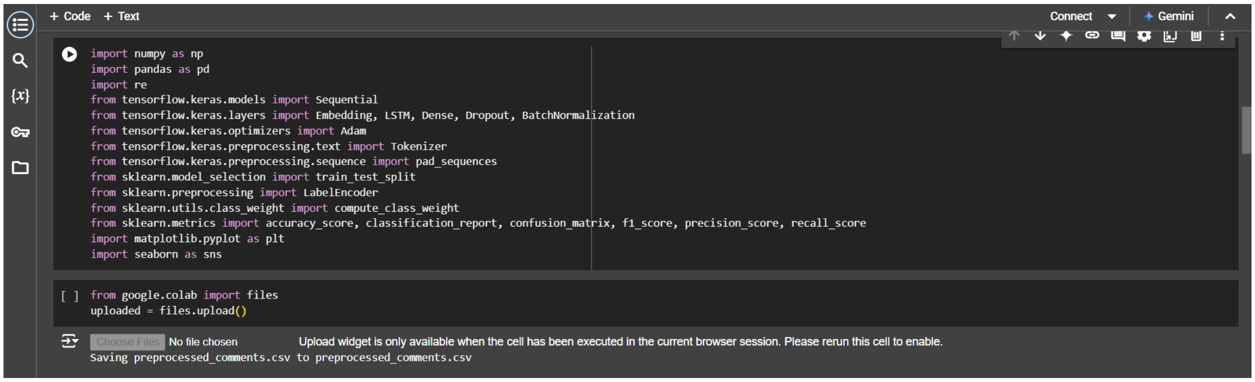
MILESTONE 3 : Hybrid Deep Learning Model Architecture

LSTM (LONG SHORT TERM MEMORY NETWORKS)



---

1. Install Required Libraries

CODE: pip install tensorflow numpy matplotlib scikit-learn

- Installs essential libraries like TensorFlow (for the LSTM model), NumPy, Matplotlib, and scikit-learn.

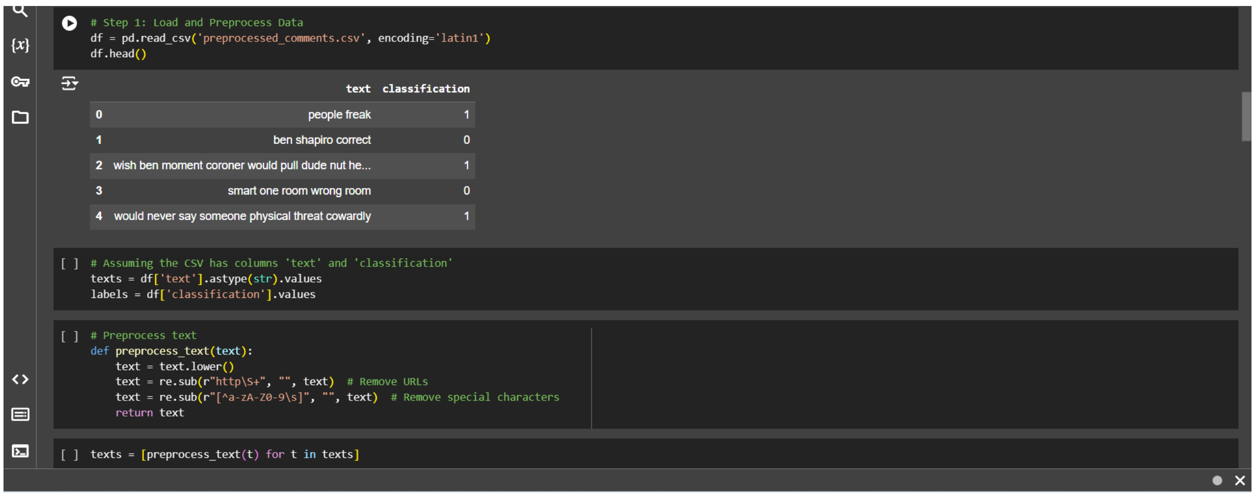
---

2. Import Libraries

CODE:

```
import numpy as np
import pandas as pd
import re
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, f1_score, precision_score, recall_score
import matplotlib.pyplot as plt
import seaborn as sns
```

- Imports libraries for data manipulation (`numpy`, `pandas`), text preprocessing (`re`), model building (`tensorflow.keras`), evaluation metrics (`sklearn`), and visualization (`matplotlib`, `seaborn`).



```
# Step 1: Load and Preprocess Data
df = pd.read_csv('preprocessed_comments.csv', encoding='latin1')
df.head()
```

	text	classification
0	people freak	1
1	ben shapiro correct	0
2	wish ben moment coroner would pull dude nut he...	1
3	smart one room wrong room	0
4	would never say someone physical threat cowardly	1

```
[ ] # Assuming the CSV has columns 'text' and 'classification'
    texts = df['text'].astype(str).values
    labels = df['classification'].values
```

```
[ ] # Preprocess text
    def preprocess_text(text):
        text = text.lower()
        text = re.sub(r"http\S+", "", text)  # Remove URLs
        text = re.sub(r"[^a-zA-Z0-9\s]", "", text)  # Remove special characters
        return text
```

```
[ ] texts = [preprocess_text(t) for t in texts]
```

3. **<u>Upload and Read Data</u>**

<u>CODE:</u>

```
from google.colab import files
uploaded = files.upload()
df = pd.read_csv('preprocessed_comments.csv', encoding='latin1')
```

- Uses Google Colab's file uploader to upload the `preprocessed_comments.csv` file.
- Reads the uploaded CSV into a Pandas DataFrame (`df`) for processing.

---

4. **<u>Extract Text and Labels</u>**

<u>CODE:</u>

```
texts = df['text'].astype(str).values
labels = df['classification'].values
```

- Extracts the `text` column (comments) and `classification` column (labels) from the DataFrame.

---

5. Preprocess Text

CODE:

```
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r"http\S+", "", text)  # Remove URLs
    text = re.sub(r"[^a-zA-Z0-9\s]", "", text)  # Remove special characters
    return text

texts = [preprocess_text(t) for t in texts]
```

- Defines a function to:
    - Convert text to lowercase.
    - Remove URLs using a regex pattern.
    - Remove non-alphanumeric characters.
- Applies this preprocessing function to all comments.

6. Encode Labels

CODE:

```
label_encoder = LabelEncoder()
labels = label_encoder.fit_transform(labels)
```

- Encodes the text labels into numerical values using `LabelEncoder` for model compatibility.

---

7. **Tokenize and Pad Sequences**

CODE:

```
max_words = 10000
max_len = 100

tokenizer = Tokenizer(num_words=max_words, oov_token="<OOV>")
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
pre_padded_sequences = pad_sequences(sequences, maxlen=max_len,
padding='pre', truncating='pre')
```

- Limits the vocabulary to 10,000 words (`max_words`) and sequences to 100 words (`max_len`).
- Uses a tokenizer to:
  - Replace out-of-vocabulary words with `<OOV>`.
  - Convert each comment into a sequence of integers.
- Pads sequences to ensure uniform length using pre-padding.

---

8. **Split Data**

CODE:

```
X_train_pre, X_test_pre, y_train_pre, y_test_pre = train_test_split(
    pre_padded_sequences, labels, test_size=0.33, random_state=42
)
```

- Splits the data into training (67%) and testing (33%) sets using `train_test_split`.

---

9. **Compute Class Weights**

CODE:

```
class_weights = compute_class_weight('balanced',
classes=np.unique(y_train_pre), y=y_train_pre)
class_weights = dict(enumerate(class_weights))
```

- Computes weights for classes to address class imbalance using `compute_class_weight`.

- Converts the result into a dictionary for model compatibility.

```
# Step 2: Build the Improved LSTM Model
lstm_model_pre = Sequential([
    Embedding(input_dim=max_words, output_dim=64),
    LSTM(64, activation='tanh', return_sequences=True),
    BatchNormalization(),
    Dropout(0.5),
    LSTM(32, activation='tanh', return_sequences=False),
    Dropout(0.5),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')  # Binary classification
])

# Compile the model
lstm_model_pre.compile(optimizer=Adam(learning_rate=0.0005), loss='binary_crossentropy', metrics=['accuracy'])
```

```
# Step 3: Train the Model
from tensorflow.keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

history_pre = lstm_model_pre.fit(
    X_train_pre, y_train_pre,
    epochs=20,
    batch_size=64,
    validation_data=(X_test_pre, y_test_pre),
    callbacks=[early_stopping],
    class_weight=class_weights
)
```

10. <u>Build LSTM Model</u>

<u>CODE:</u>

```
lstm_model_pre = Sequential([
   Embedding(input_dim=max_words, output_dim=64),
   LSTM(64, activation='tanh', return_sequences=True),
   BatchNormalization(),
   Dropout(0.5),
   LSTM(32, activation='tanh', return_sequences=False),
   Dropout(0.5),
   Dense(16, activation='relu'),
   Dense(1, activation='sigmoid')
])
```

- Creates a sequential LSTM model with:
  - **Embedding layer**: Converts word indices into dense vectors.
  - **First LSTM layer**: Captures temporal dependencies; outputs sequences.
  - **BatchNormalization**: Stabilizes and accelerates training.
  - **Dropout**: Prevents overfitting.
  - **Second LSTM layer**: Outputs the final sequence representation.
  - **Dense layers**: Perform classification. The final layer uses `sigmoid` for binary output.

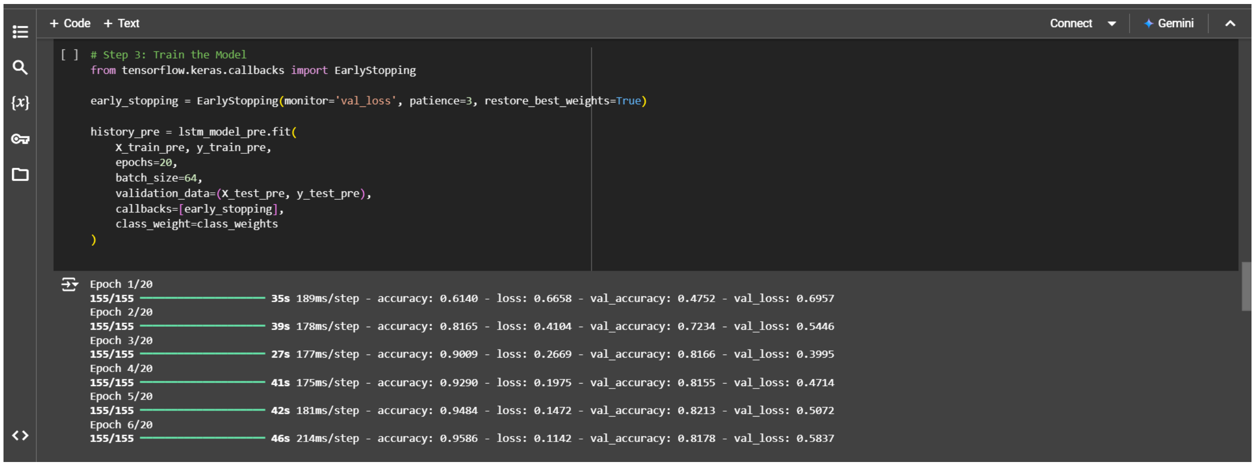
---

11. Compile the Model

CODE:

lstm_model_pre.compile(optimizer=Adam(learning_rate=0.0005), loss='binary_crossentropy', metrics=['accuracy'])

- Configures the model with:
  - `Adam` optimizer for adaptive learning.
  - `binary_crossentropy` as the loss function.
  - `accuracy` as the evaluation metric.

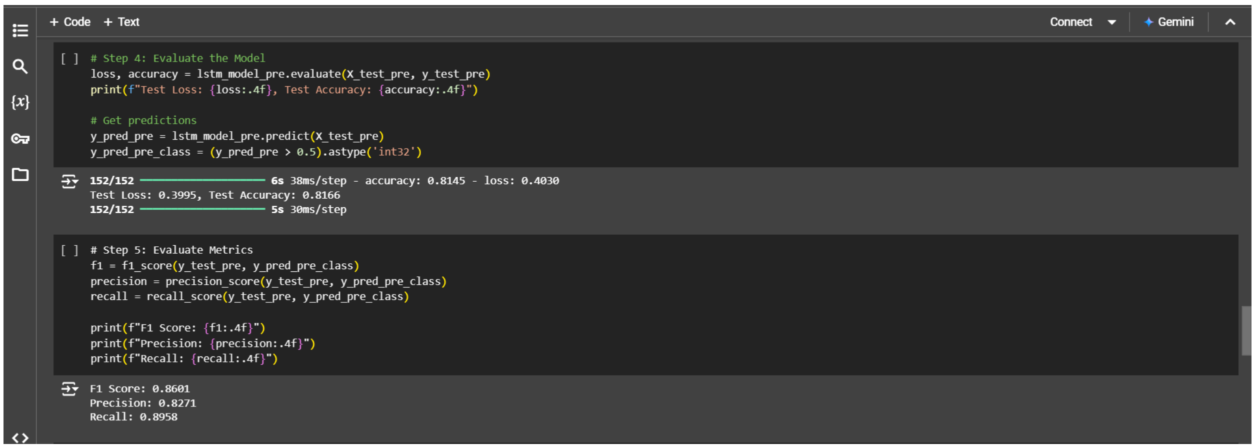


12. Train the Model

CODE:

```
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)

history_pre = lstm_model_pre.fit(
   X_train_pre, y_train_pre,
   epochs=20,
   batch_size=64,
   validation_data=(X_test_pre, y_test_pre),
   callbacks=[early_stopping],
   class_weight=class_weights )
```

- Trains the model for up to 20 epochs with:
    - Batch size of 64.
    - Early stopping to prevent overfitting by monitoring validation loss.
    - `class_weights` to handle imbalance.



13. Evaluate the Model

CODE:

```
loss, accuracy = lstm_model_pre.evaluate(X_test_pre, y_test_pre)
```

- Evaluates the trained model on the test set and prints the loss and accuracy.

---

14. Get Predictions

CODE:

```
y_pred_pre = lstm_model_pre.predict(X_test_pre)
y_pred_pre_class = (y_pred_pre > 0.5).astype('int32')
```

- Gets the model's predictions and converts them to binary classifications using a 0.5 threshold.

---

15. Compute Metrics

CODE:

```
f1 = f1_score(y_test_pre, y_pred_pre_class)
precision = precision_score(y_test_pre, y_pred_pre_class)
recall = recall_score(y_test_pre, y_pred_pre_class)
```

- Computes the F1 score, precision, and recall for model evaluation.

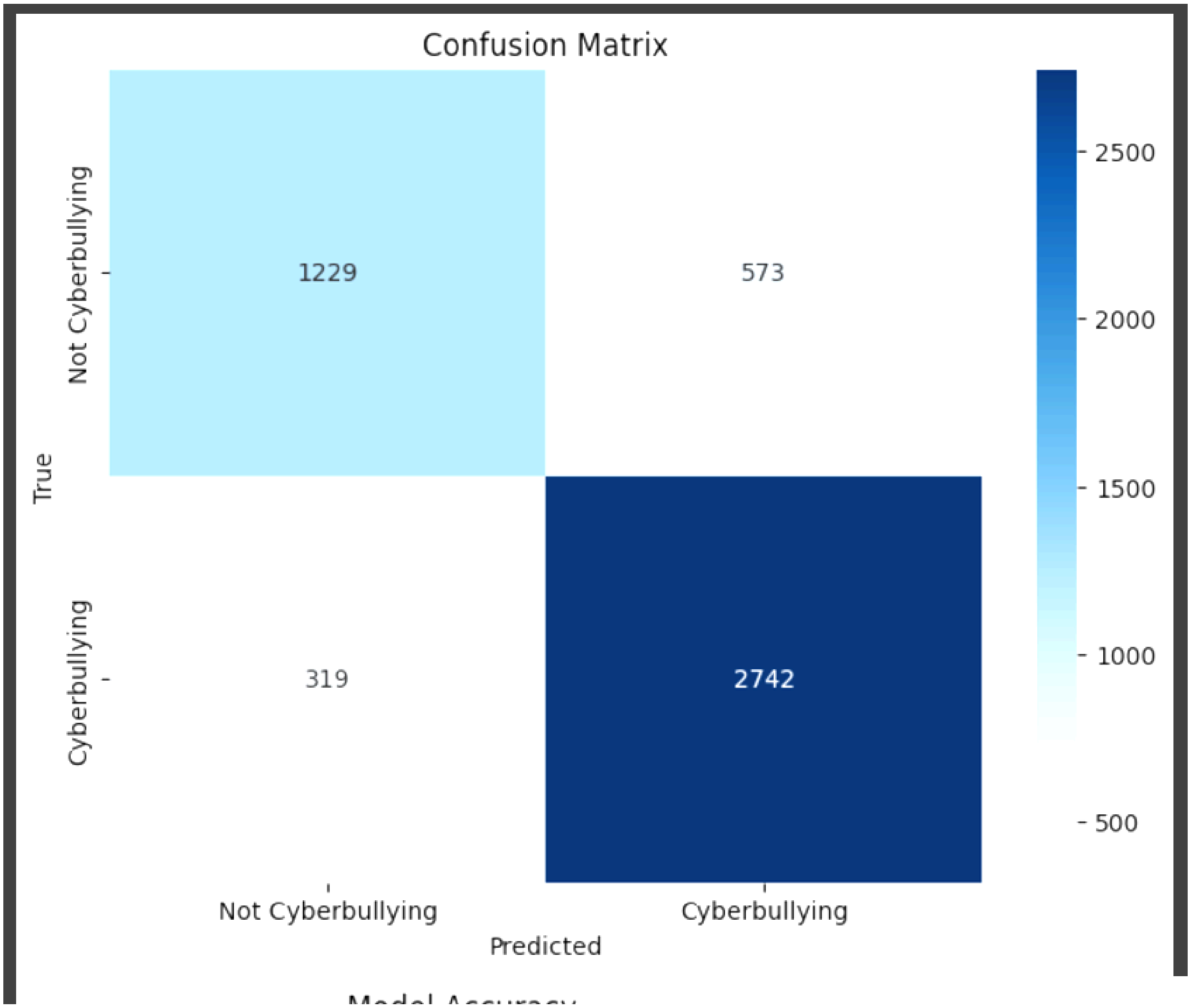
```
# Confusion Matrix
conf_matrix_pre = confusion_matrix(y_test_pre, y_pred_pre_class)
class_names = ['Not Cyberbullying', 'Cyberbullying']

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_pre, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()

# Step 6: Visualize Training History
plt.figure(figsize=(12, 5))

# Accuracy Plot
plt.subplot(1, 2, 1)
plt.plot(history_pre.history['accuracy'], label='Training Accuracy')
plt.plot(history_pre.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.legend()

# Loss Plot
plt.subplot(1, 2, 2)
plt.plot(history_pre.history['loss'], label='Training Loss')
plt.plot(history_pre.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.legend()

plt.show()
```

16. <u>Confusion Matrix</u>

<u>CODE:</u>

```
conf_matrix_pre = confusion_matrix(y_test_pre, y_pred_pre_class)
class_names = ['Not Cyberbullying', 'Cyberbullying']

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_pre, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

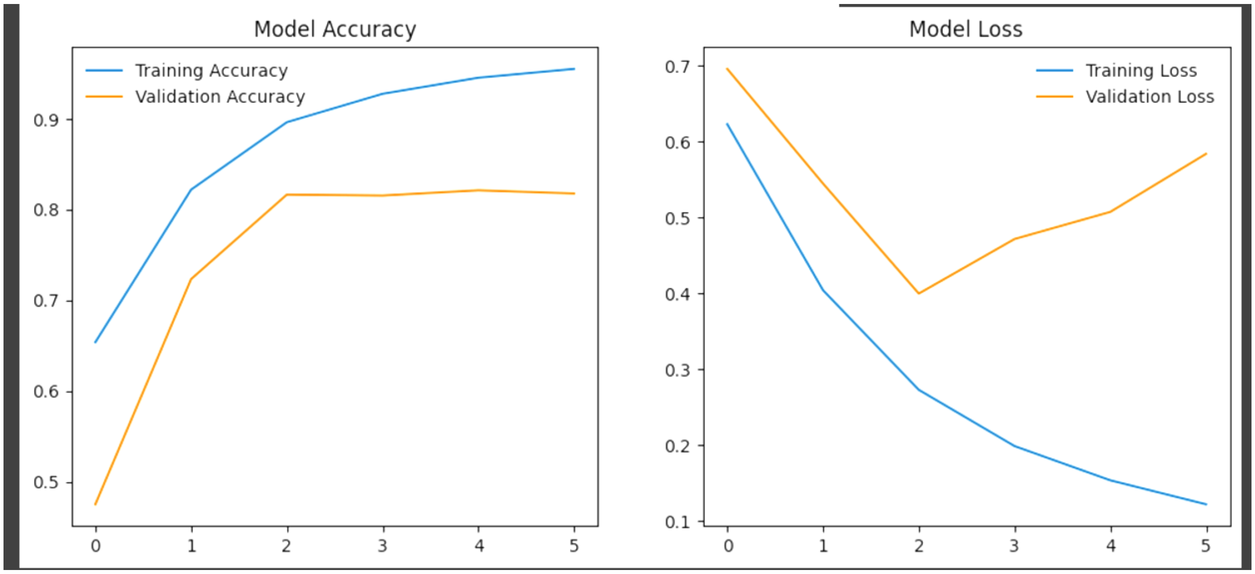
- Plots a confusion matrix to visualize the model's predictions.



17. Visualize Training History

```
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history_pre.history['accuracy'], label='Training Accuracy')
plt.plot(history_pre.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(history_pre.history['loss'], label='Training Loss')
plt.plot(history_pre.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.legend()
plt.show()
```

- Plots training and validation accuracy and loss over epochs.



---

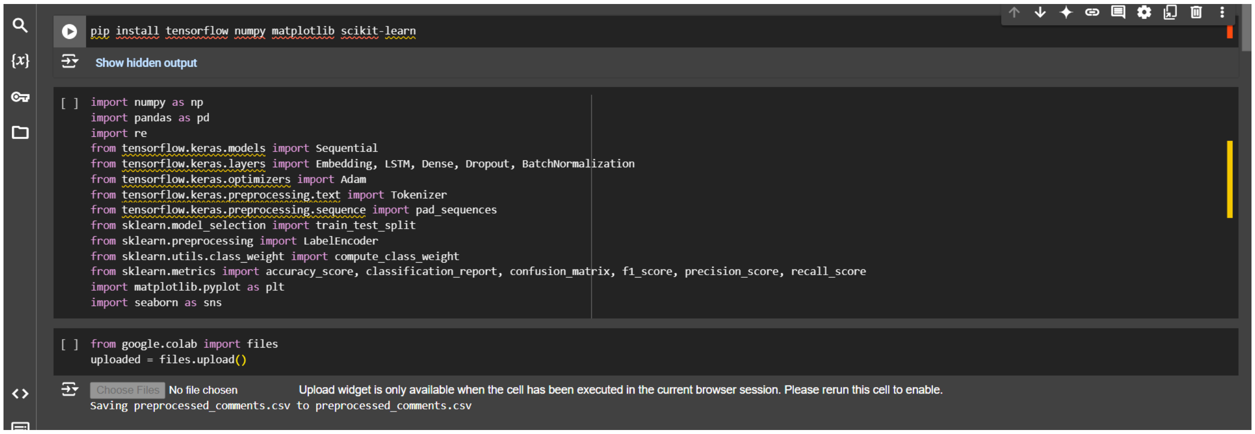
18. <u>Save and Download the Model</u>

<u>CODE:</u>

```
lstm_model_pre.save('lstm_model.h5')
files.download('lstm_model.h5')
```

- Saves the trained model to a `.h5` file and downloads it.

RNN (Recurrent Neural Network)



---

1. **Setup and Libraries**

   CODE:pip install tensorflow numpy matplotlib scikit-learn

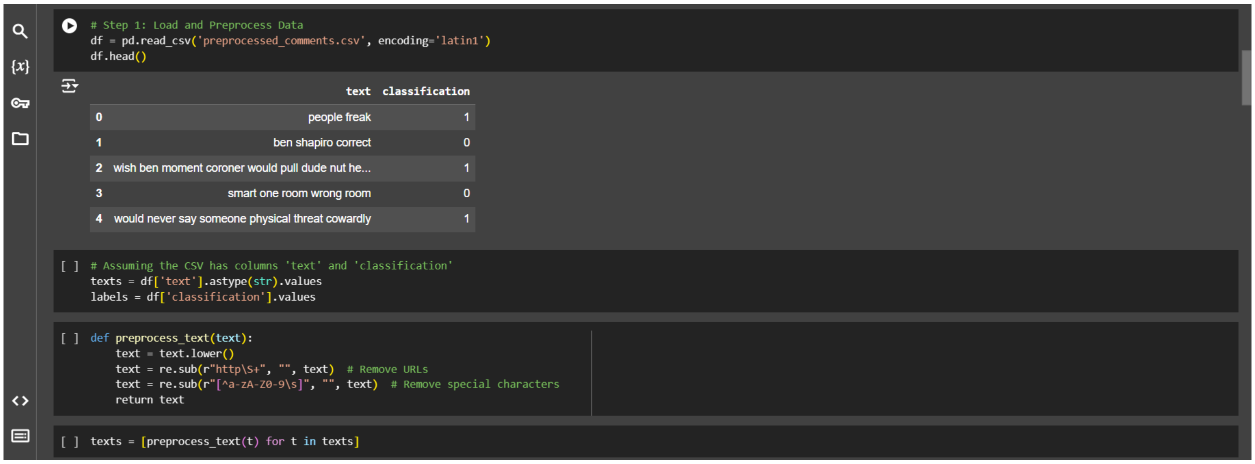
- Installs the required libraries: TensorFlow (for deep learning), NumPy (for numerical operations), Matplotlib (for visualization), and scikit-learn (for machine learning utilities).

```
import numpy as np
import pandas as pd
import re
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.utils.class_weight import compute_class_weight
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, f1_score, precision_score, recall_score
import matplotlib.pyplot as plt
import seaborn as sns
```

- Imports libraries and modules required for preprocessing, deep learning, model evaluation, and visualization.

  CODE: from google.colab import files
  uploaded = files.upload()

- Allows file upload in Google Colab. Users upload a CSV file for processing.



2. **Step 1: Load and Preprocess Data**

   CODE: df = pd.read_csv('preprocessed_comments.csv', encoding='latin1')
   df.head()

- Loads the dataset into a Pandas DataFrame and displays the first few rows. Assumes the file has a column `text` (for input) and `classification` (for labels).

  CODE:
  texts = df['text'].astype(str).values
  labels = df['classification'].values

- Extracts the text and classification columns as NumPy arrays.

  CODE:
```
def preprocess_text(text):
    text = text.lower()  # Converts to lowercase.
    text = re.sub(r"http\S+", "", text)  # Removes URLs.
    text = re.sub(r"[^a-zA-Z0-9\s]", "", text)  # Removes special characters.
    return text
texts = [preprocess_text(t) for t in texts]
```

- Defines a function to preprocess text data by lowercasing, removing URLs, and cleaning special characters. Applies the function to all texts.

```
[ ] # Encode labels
    label_encoder = LabelEncoder()
    labels = label_encoder.fit_transform(labels)

    # Tokenize and pad the text data
    max_words = 10000  # Maximum vocabulary size
    max_len = 100      # Maximum sequence length

    tokenizer = Tokenizer(num_words=max_words, oov_token="<OOV>")
    tokenizer.fit_on_texts(texts)
    sequences = tokenizer.texts_to_sequences(texts)
    post_padded_sequences = pad_sequences(sequences, maxlen=max_len, padding='pre', truncating='post')

[ ] # Split data into training and testing sets
    X_train_pre, X_test_pre, y_train_pre, y_test_pre = train_test_split(
        post_padded_sequences, labels, test_size=0.33, random_state=42
    )

[ ] # Compute class weights
    class_weights = compute_class_weight('balanced', classes=np.unique(y_train_pre), y=y_train_pre)
    class_weights = dict(enumerate(class_weights))
```

<u>CODE:</u>

```
label_encoder = LabelEncoder()
labels = label_encoder.fit_transform(labels)
```

- Encodes the class labels into integers for model compatibility (e.g., `positive`, `negative` → `1`, `0`).

<u>CODE:</u>

```
max_words = 10000  # Maximum vocabulary size
max_len = 100      # Maximum sequence length
```

- Specifies tokenizer vocabulary size and maximum sequence length.

<u>CODE:</u>

```
tokenizer = Tokenizer(num_words=max_words, oov_token="<OOV>")
tokenizer.fit_on_texts(texts)
sequences = tokenizer.texts_to_sequences(texts)
post_padded_sequences = pad_sequences(sequences, maxlen=max_len, padding='pre', truncating='post')
```

- Initializes a tokenizer, converts texts to sequences of integers (word IDs), and pads/truncates sequences to a fixed length.

CODE:

```
X_train_pre, X_test_pre, y_train_pre, y_test_pre = train_test_split(
    post_padded_sequences, labels, test_size=0.33, random_state=42
)
```

- Splits data into training and testing sets (33% for testing).

CODE:

```
class_weights = compute_class_weight('balanced', classes=np.unique(y_train_pre), y=y_train_pre)
class_weights = dict(enumerate(class_weights))
```

- Computes class weights to handle class imbalance and converts them into a dictionary for use in training.

```
[ ] # Step 2: Build the Improved RNN Model
rnn_pre = Sequential([
    Embedding(input_dim=max_words, output_dim=64),
    LSTM(64, activation='tanh', return_sequences=True),
    BatchNormalization(),
    Dropout(0.5),
    LSTM(32, activation='tanh', return_sequences=False),
    Dropout(0.5),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')
])

# Compile the model
rnn_pre.compile(optimizer=Adam(learning_rate=0.0005), loss='binary_crossentropy', metrics=['accuracy'])
```

3. **Step 2: Build the RNN Model**

CODE:

```
rnn_pre = Sequential([
    Embedding(input_dim=max_words, output_dim=64),
    LSTM(64, activation='tanh', return_sequences=True),
    BatchNormalization(),
    Dropout(0.5),
    LSTM(32, activation='tanh', return_sequences=False),
    Dropout(0.5),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')
])
```

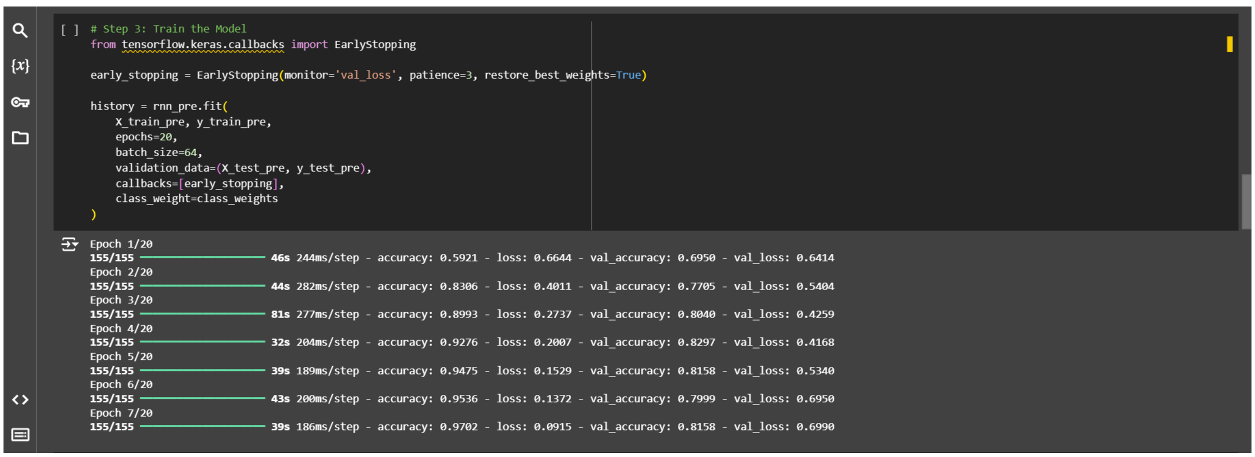
- Defines an RNN model:
    - **Embedding**: Converts word IDs into dense vectors of fixed size.
    - **LSTM**: Long Short-Term Memory layers to capture sequential dependencies.
    - **BatchNormalization**: Normalizes activations for faster convergence.

- **Dropout**: Reduces overfitting by randomly dropping connections.
- **Dense**: Fully connected layers, final layer outputs a single value with a sigmoid activation for binary classification.

CODE:

```
rnn_pre.compile(optimizer=Adam(learning_rate=0.0005),
loss='binary_crossentropy', metrics=['accuracy'])
```

- Compiles the model with:
  - **Adam** optimizer for gradient updates.
  - **Binary Cross-Entropy** loss for binary classification.
  - **Accuracy** as a metric for monitoring performance.



4. **Step 3: Train the Model**

CODE:

```
from tensorflow.keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)
```

- Sets up early stopping to terminate training if validation loss does not improve for 3 epochs, restoring the best weights.

CODE:

```
history = rnn_pre.fit(
    X_train_pre, y_train_pre,
    epochs=20,
    batch_size=64,
    validation_data=(X_test_pre, y_test_pre),
    callbacks=[early_stopping],
    class_weight=class_weights
)
```

- Trains the model for 20 epochs with:
  - **Batch size** of 64.
  - Validation on the test set.
  - Early stopping and class weights applied.

5. **Step 4: Evaluate the Model**

CODE:

```
loss, accuracy = rnn_pre.evaluate(X_test_pre, y_test_pre)
print(f"Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}")
```

- Evaluates the trained model on the test set and prints loss and accuracy.

CODE:

```
y_pred_pre = rnn_pre.predict(X_test_pre)
y_pred_pre_class = (y_pred_pre > 0.5).astype('int32')
```

- Makes predictions and converts probabilities to binary class labels using a threshold of 0.5.

---

6. **Step 5: Evaluate Metrics**

CODE:

```
f1 = f1_score(y_test_pre, y_pred_pre_class)
precision = precision_score(y_test_pre, y_pred_pre_class)
recall = recall_score(y_test_pre, y_pred_pre_class)
print(f"F1 Score: {f1:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
```

- Calculates and prints F1 Score, Precision, and Recall for performance evaluation.

CODE:

```
conf_matrix = confusion_matrix(y_test_pre, y_pred_pre_class)
class_names = ['Class 0', 'Class 1']
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

- Computes and visualizes the confusion matrix using Seaborn.

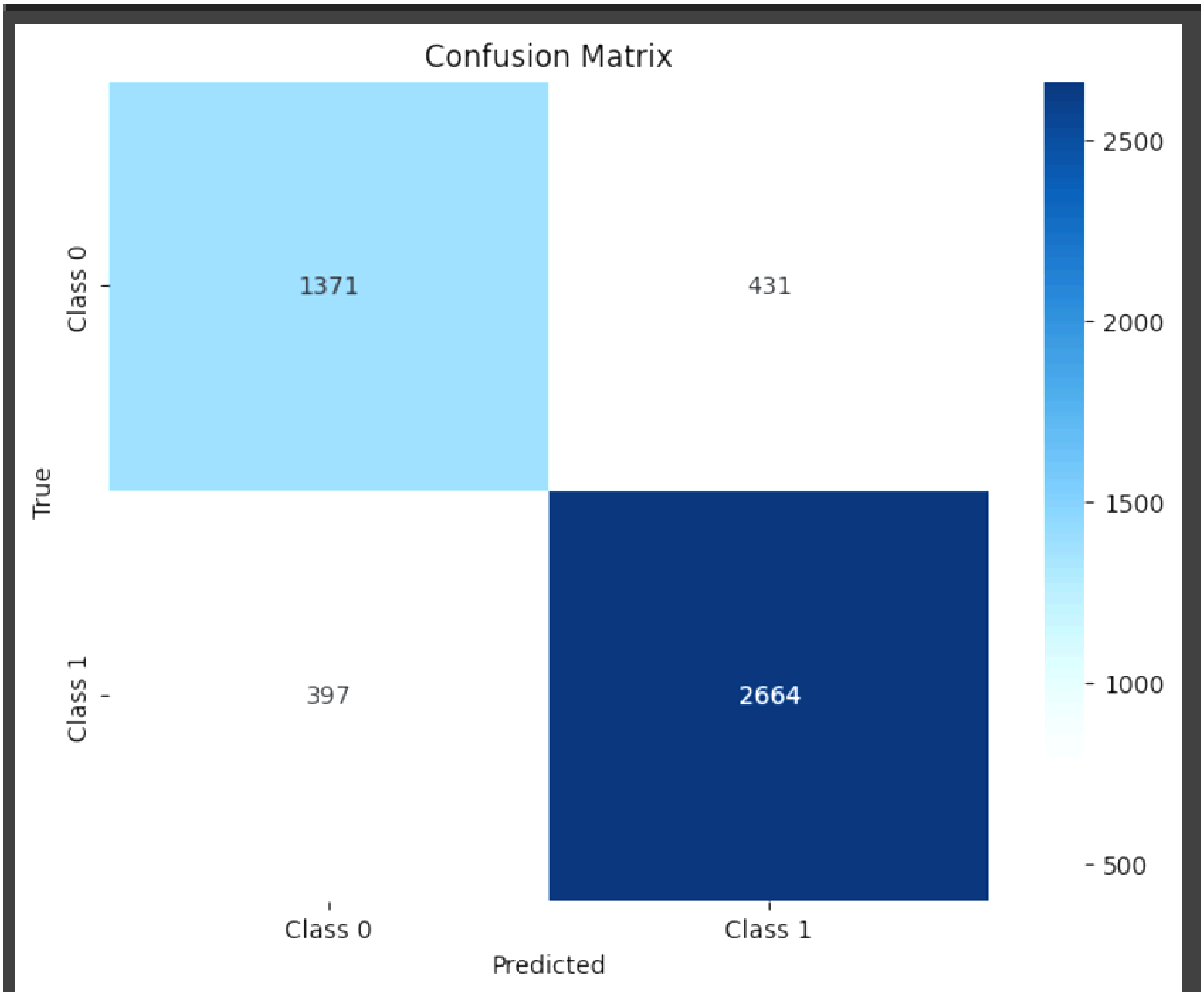
---

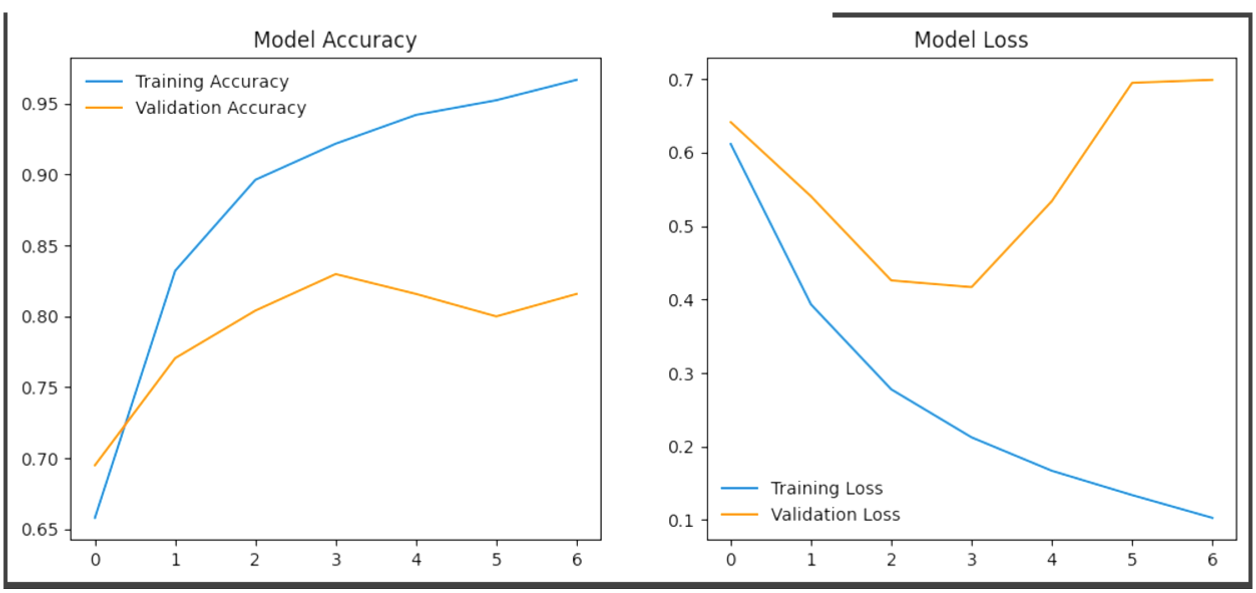
7. **Step 6: Visualize Training History**

CODE:

```
plt.figure(figsize=(12, 5))

# Accuracy Plot
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.legend()

# Loss Plot
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.legend()

plt.show()
```

- Visualizes training and validation accuracy and loss over epochs.


Confusion Matrix
1371
431
397
2664
Class 0
Class 1
Class 0
Class 1
True
Predicted
2500
2000
1500
1000
500


Model Accuracy
Training Accuracy
Validation Accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0
1
2
3
4
5
6
Model Loss
Training Loss
Validation Loss
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1
2
3
4
5
6

```
[ ] rnn_pre.save('rnn_model.h5')
    files.download('rnn_model.h5')
    import pickle

    # Save the tokenizer to a file
    with open('tokenizer.pkl', 'wb') as f:
        pickle.dump(tokenizer, f)
    files.download('tokenizer.pkl')
```

8. Save Model and Tokenizer

CODE:

```
rnn_pre.save('rnn_model.h5')
files.download('rnn_model.h5')
```

- Saves the trained model as `rnn_model.h5` and allows downloading.

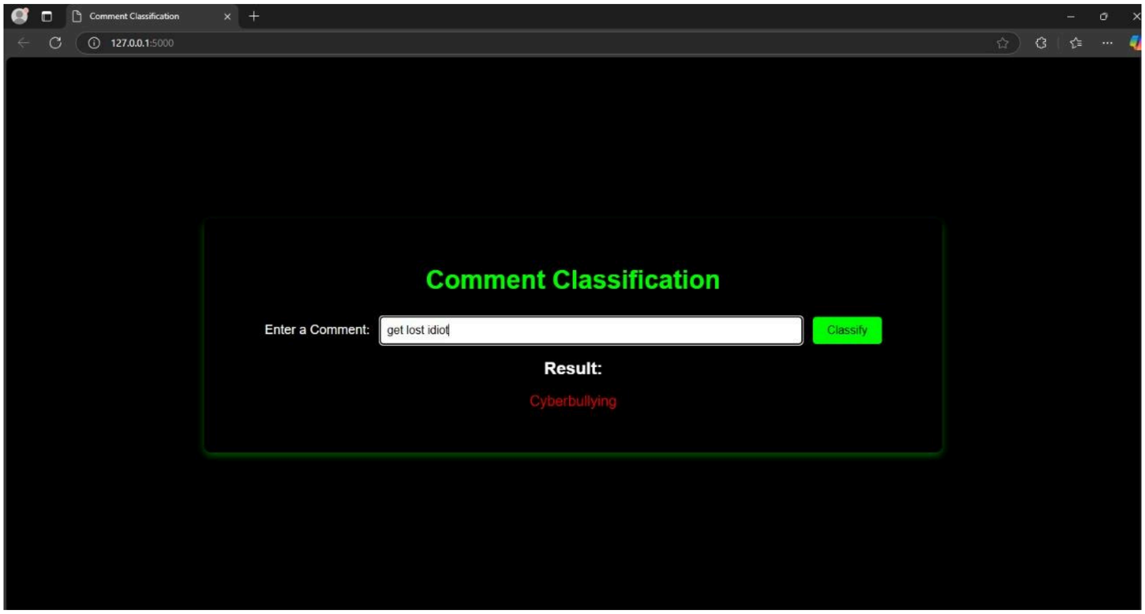
CODE:

```
import pickle
with open('tokenizer.pkl', 'wb') as f:
    pickle.dump(tokenizer, f)
files.download('tokenizer.pkl')
```

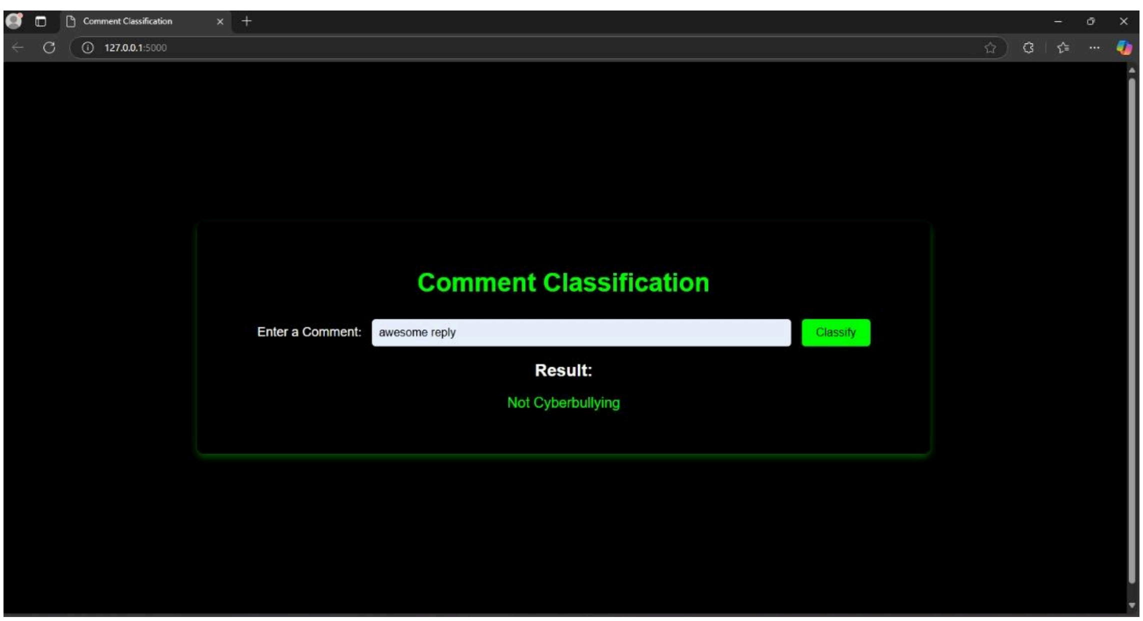
- Serializes and saves the tokenizer as `tokenizer.pkl`, then allows downloading.

Front End GUI

With Negative Comment:“ GET LOST IDIOT ”

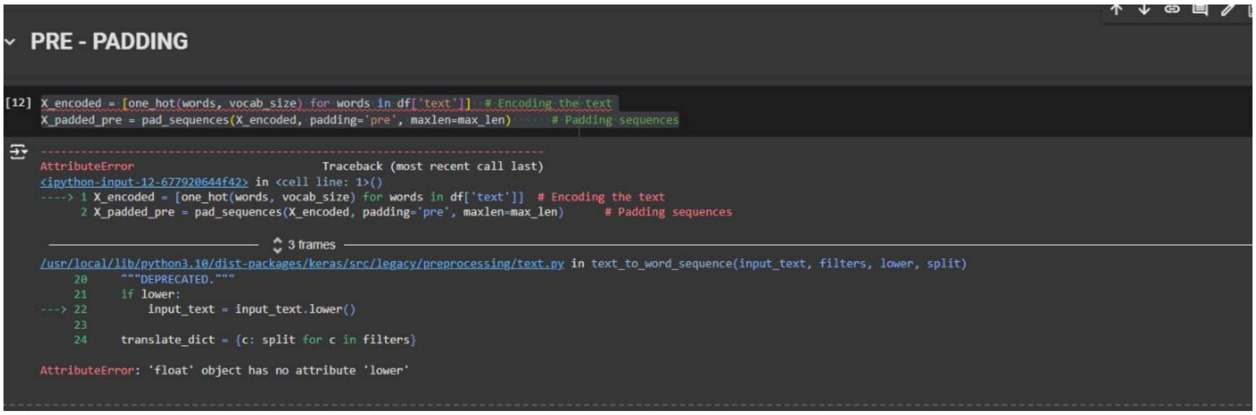


With Positive Comment:“AWESOME REPLY”



ERRORS ENCOUNTERED

ERROR 1:



PRE - PADDING

```
[12] X_encoded = [one_hot(words, vocab_size) for words in df['text']]  # Encoding the text
     X_padded_pre = pad_sequences(X_encoded, padding='pre', maxlen=max_len)      # Padding sequences

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-12-677920644f42> in <cell line: 1>()
----> 1 X_encoded = [one_hot(words, vocab_size) for words in df['text']]  # Encoding the text
      2 X_padded_pre = pad_sequences(X_encoded, padding='pre', maxlen=max_len)      # Padding sequences

    3 frames
/usr/local/lib/python3.10/dist-packages/keras/src/legacy/preprocessing/text.py in text_to_word_sequence(input_text, filters, lower, split)
     20     """DEPRECATED."""
     21     if lower:
---> 22         input_text = input_text.lower()
     23 
     24     translate_dict = {c: split for c in filters}

AttributeError: 'float' object has no attribute 'lower'
```

REASON:

The error you encountered, `float object has no attribute`, often arises when you mistakenly treat a float (numeric value) as if it were an object with attributes (e.g., calling a method like `.shape` or `.fit` on a float)

****

ERROR 2:

ValueError Traceback (most recent call last) <ipython-input-14-31a443235a59> in <cell line: 2>() 1 # Train the model ----> 2 history = lstm.fit(X_train, y_train_categorical, epochs=10, batch_size=64, validation_data=(X_test, y_test_categorical)) 1 frames /usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/__init__.py in get_data_adapter(x, y, sample_weight, batch_size, steps_per_epoch, shuffle, class_weight) 118 #) 119 else: --> 120 raise ValueError(f"Unrecognized data type: x={x} (of type {type(x)})") 121 122

REASON:

The error occurs because the data provided to the `lstm.fit()` method is not in a format that Keras recognizes for training. The issue lies in the format of `X_train`, which appears to be a list of lists or an improperly prepared NumPy array. For training an LSTM model, Keras expects the input data (`X_train`) to be a 3D array of shape `(samples, timesteps, features)`, where:

- `samples`: Number of training examples.
- `timesteps`: Number of time steps in each sequence.
- `features`: Number of features per time step.