Experiment no 3:

Name:Jhanvi Parekh

Sap:60009210033

Lab exercise to be performed:

Consider fake news data set

Data Preprocessing

1. Load the Dataset
2. Text Cleaning:

   Clean the text data (remove special characters, convert to lowercase, etc.).
3. Tokenization and Padding:

   Tokenize and pad the text sequences.
4. Split Dataset:

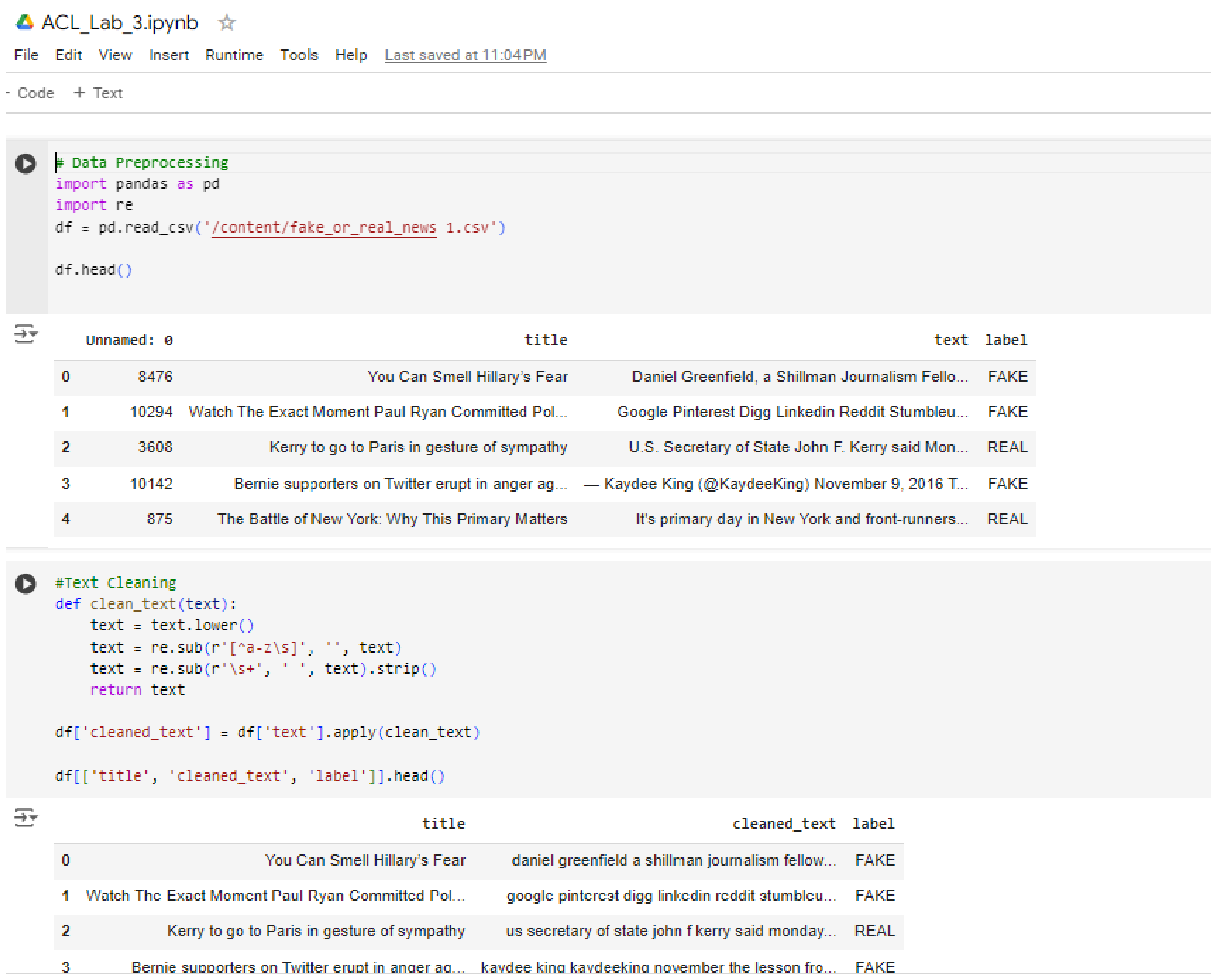
   Split the data into training and test sets.

2.Building the RNN Model

3. Building the LSTM Model

4. Building GRU Model

5. Perform Fake News Classification for English News

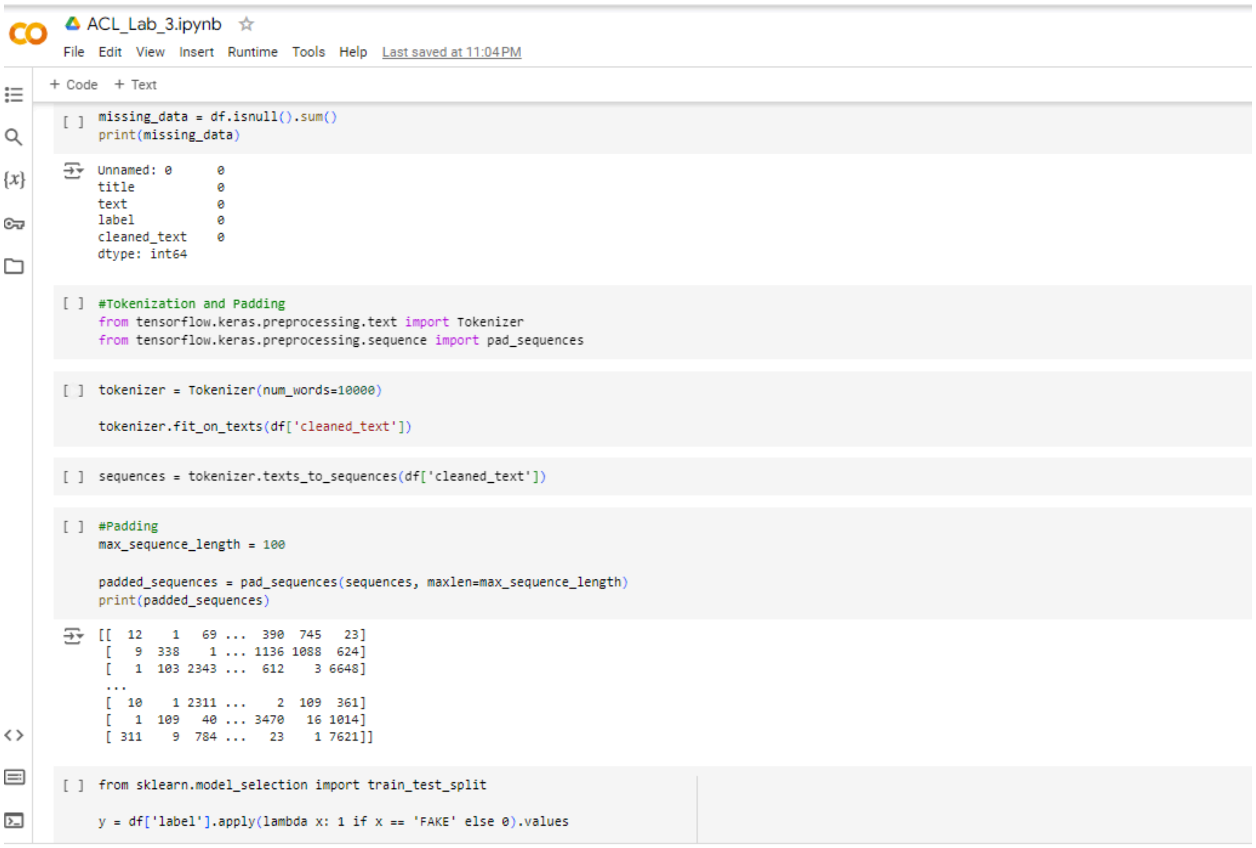

ACL_Lab_3.ipynb
File Edit View Insert Runtime Tools Help Last saved at 11:04 PM
Code + Text

```python
# Data Preprocessing
import pandas as pd
import re
df = pd.read_csv('/content/fake_or_real_news 1.csv')

df.head()
```

```
   Unnamed: 0                                              title                                             text  label
0        8476                       You Can Smell Hillary's Fear  Daniel Greenfield, a Shillman Journalism Fello...  FAKE
1       10294  Watch The Exact Moment Paul Ryan Committed Pol...  Google Pinterest Digg Linkedin Reddit Stumbleu...  FAKE
2        3608              Kerry to go to Paris in gesture of sympathy  U.S. Secretary of State John F. Kerry said Mon...  REAL
3       10142  Bernie supporters on Twitter erupt in anger ag...  — Kaydee King (@KaydeeKing) November 9, 2016 T...  FAKE
4         875           The Battle of New York: Why This Primary Matters  It's primary day in New York and front-runners...  REAL
```

```python
#Text Cleaning
def clean_text(text):
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)
    text = re.sub(r'\s+', ' ', text).strip()
    return text

df['cleaned_text'] = df['text'].apply(clean_text)

df[['title', 'cleaned_text', 'label']].head()
```

```
                                                title                                      cleaned_text  label
0                        You Can Smell Hillary's Fear  daniel greenfield a shillman journalism fellow...  FAKE
1  Watch The Exact Moment Paul Ryan Committed Pol...  google pinterest digg linkedin reddit stumbleu...  FAKE
2         Kerry to go to Paris in gesture of sympathy  us secretary of state john f kerry said monday...  REAL
3   Bernie supporters on Twitter erupt in anger ag...  kaydee king kaydeeking november the lesson fro...  FAKE
```


ACL_Lab_3.ipynb
File Edit View Insert Runtime Tools Help Last saved at 11:04 PM
+ Code + Text

```
missing_data = df.isnull().sum()
print(missing_data)
```

```
Unnamed: 0      0
title           0
text            0
label           0
cleaned_text    0
dtype: int64
```

```
#Tokenization and Padding
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

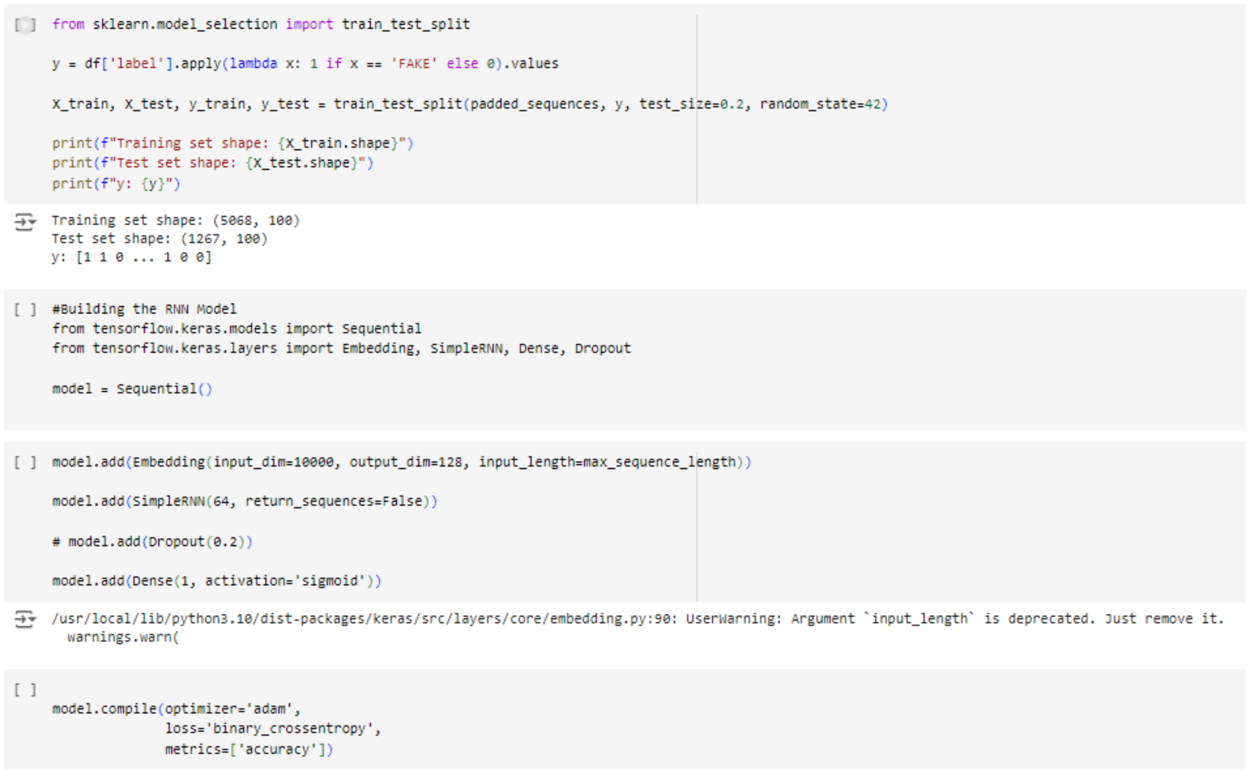
```
tokenizer = Tokenizer(num_words=10000)

tokenizer.fit_on_texts(df['cleaned_text'])
```

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

```
#Padding
max_sequence_length = 100

padded_sequences = pad_sequences(sequences, maxlen=max_sequence_length)
print(padded_sequences)
```

```
[[  12    1   69 ...  390  745   23]
 [   9  338    1 ... 1136 1088  624]
 [   1  103 2343 ...  612    3 6648]
 ...
 [  10    1 2311 ...    2  109  361]
 [   1  109   40 ... 3470   16 1014]
 [ 311    9  784 ...   23    1 7621]]
```

```
from sklearn.model_selection import train_test_split

y = df['label'].apply(lambda x: 1 if x == 'FAKE' else 0).values
```



```
from sklearn.model_selection import train_test_split

y = df['label'].apply(lambda x: 1 if x == 'FAKE' else 0).values

X_train, X_test, y_train, y_test = train_test_split(padded_sequences, y, test_size=0.2, random_state=42)

print(f"Training set shape: {X_train.shape}")
print(f"Test set shape: {X_test.shape}")
print(f"y: {y}")
```

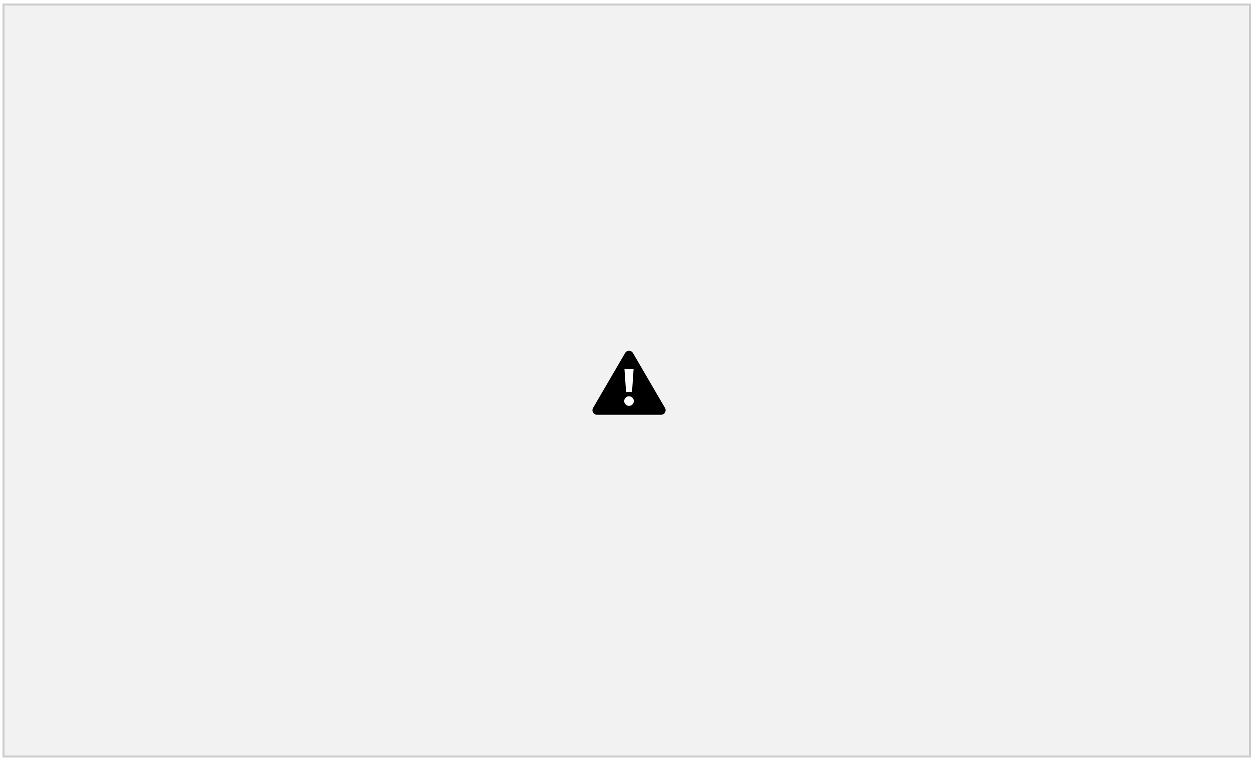
```
Training set shape: (5068, 100)
Test set shape: (1267, 100)
y: [1 1 0 ... 1 0 0]
```

```
#Building the RNN Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense, Dropout

model = Sequential()
```

```
model.add(Embedding(input_dim=10000, output_dim=128, input_length=max_sequence_length))

model.add(SimpleRNN(64, return_sequences=False))

# model.add(Dropout(0.2))

model.add(Dense(1, activation='sigmoid'))
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/core/embedding.py:90: UserWarning: Argument `input_length` is deprecated. Just remove it.
  warnings.warn(
```

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



```
[ ] predictions = model.predict(X_test)

    predicted_labels = (predictions > 0.5).astype(int)
    print("Predicted Labels:")
    print(predicted_labels)
```

```
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step
Predicted Labels:
[[1]
 [1]
 [1]
 ...
 [0]
 [0]
 [1]]
```

```
[ ] for i in range(10):
        print(f"Sample {i+1}")
        print(f"Actual Label: {'FAKE' if y_test[i] == 1 else 'REAL'}")
        print(f"Predicted Label: {'FAKE' if predicted_labels[i] == 1 else 'REAL'}")
        print("-" * 30)
```

```
Sample 1
Actual Label: FAKE
Predicted Label: FAKE
------------------------------
Sample 2
Actual Label: FAKE
Predicted Label: FAKE
------------------------------
Sample 3
Actual Label: FAKE
Predicted Label: FAKE
------------------------------
Sample 4
Actual Label: FAKE
```

```
------------------------------
Sample 6
Actual Label: FAKE
Predicted Label: FAKE
------------------------------
Sample 7
Actual Label: REAL
Predicted Label: REAL
------------------------------
Sample 8
Actual Label: FAKE
Predicted Label: FAKE
------------------------------
Sample 9
Actual Label: REAL
Predicted Label: REAL
------------------------------
Sample 10
Actual Label: FAKE
Predicted Label: FAKE
------------------------------
```

```
[ ] #Building the LSTM Model
    from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
```

```
[ ] model1 = Sequential()

    model1.add(Embedding(input_dim=10000, output_dim=128, input_length=max_sequence_length))

    model1.add(LSTM(64, return_sequences=False))

    model1.add(Dropout(0.5))

    model1.add(Dense(1, activation='sigmoid'))
```

```
[ ] model1.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

```
[ ] model1.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

```
history1 = model1.fit(X_train, y_train,
                      epochs=5,
                      batch_size=64,
                      validation_data=(X_test, y_test))
```

```
Epoch 1/5
80/80 ━━━━━━━━━━━━━━━━━━━━ 14s 150ms/step - accuracy: 0.6191 - loss: 0.6372 - val_accuracy: 0.8579 - val_loss: 0.3389
Epoch 2/5
80/80 ━━━━━━━━━━━━━━━━━━━━ 12s 144ms/step - accuracy: 0.9111 - loss: 0.2452 - val_accuracy: 0.8437 - val_loss: 0.3431
Epoch 3/5
80/80 ━━━━━━━━━━━━━━━━━━━━ 12s 147ms/step - accuracy: 0.9615 - loss: 0.1319 - val_accuracy: 0.8650 - val_loss: 0.3925
Epoch 4/5
80/80 ━━━━━━━━━━━━━━━━━━━━ 12s 149ms/step - accuracy: 0.9861 - loss: 0.0486 - val_accuracy: 0.8556 - val_loss: 0.4358
Epoch 5/5
80/80 ━━━━━━━━━━━━━━━━━━━━ 20s 143ms/step - accuracy: 0.9933 - loss: 0.0236 - val_accuracy: 0.8382 - val_loss: 0.6501
```

```
[ ] loss, accuracy = model1.evaluate(X_test, y_test)
    print(f"Test Loss: {loss}")
    print(f"Test Accuracy: {accuracy}")
```

```
40/40 ━━━━━━━━━━━━━━━━━━━━ 1s 17ms/step - accuracy: 0.8320 - loss: 0.6778
Test Loss: 0.6500509977340698
Test Accuracy: 0.8382004499435425
```

```
[ ] # Building GRU Model
    from tensorflow.keras.layers import Embedding, GRU, Dense, Dropout
```

```
[ ] model2 = Sequential()

    model2.add(Embedding(input_dim=10000, output_dim=128, input_length=max_sequence_length))

    model2.add(GRU(64, return_sequences=False))

    model2.add(Dropout(0.5))

    model2.add(Dense(1, activation='sigmoid'))
```

```
[ ] model2.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

```
[ ] history2 = model2.fit(X_train, y_train,
                          epochs=5,
                          batch_size=64,
                          validation_data=(X_test, y_test))
```

```
Epoch 1/5
80/80 ━━━━━━━━━━━━━━━━━━━━ 15s 144ms/step - accuracy: 0.6283 - loss: 0.6456 - val_accuracy: 0.8145 - val_loss: 0.4055
Epoch 2/5
80/80 ━━━━━━━━━━━━━━━━━━━━ 22s 161ms/step - accuracy: 0.8938 - loss: 0.2623 - val_accuracy: 0.8382 - val_loss: 0.3697
Epoch 3/5
80/80 ━━━━━━━━━━━━━━━━━━━━ 20s 153ms/step - accuracy: 0.9656 - loss: 0.1104 - val_accuracy: 0.8540 - val_loss: 0.4207
Epoch 4/5
80/80 ━━━━━━━━━━━━━━━━━━━━ 21s 155ms/step - accuracy: 0.9829 - loss: 0.0582 - val_accuracy: 0.8390 - val_loss: 0.4808
Epoch 5/5
80/80 ━━━━━━━━━━━━━━━━━━━━ 21s 157ms/step - accuracy: 0.9933 - loss: 0.0384 - val_accuracy: 0.8421 - val_loss: 0.6642
```

```
[ ] loss, accuracy = model2.evaluate(X_test, y_test)
```

Model Comparison

RNN Model:

Advantages: Simple architecture that is easy to implement. Disadvantages: Struggles with capturing long-term dependencies and maintaining context over extended sequences. Best For: Suitable for tasks involving short sequences or where long-term context is not critical.

LSTM Model:

Advantages: Specifically designed to manage long-term dependencies through its gating mechanisms, which help preserve important information across longer sequences. Disadvantages: More complex and computationally intensive compared to a basic RNN. Best For: Ideal for tasks that require robust context retention and handling of longer sequences, such as textual data in fake news classification.

GRU Model:

Advantages: Offers similar performance to LSTM but with a more straightforward architecture and fewer parameters, which can lead to faster training and lower resource consumption. Disadvantages: Although efficient, its performance can vary depending on the dataset and specific task. Best For: Provides a good balance between performance and computational efficiency, making it suitable for scenarios where both performance and training efficiency are important.

Recommendations

If Performance is Critical: LSTM is generally the preferred choice due to its superior ability to handle long-term dependencies compared to basic RNNs. For datasets with long sequences that require deep context understanding, LSTM often delivers the best results.

If Training Efficiency is a Priority: GRU can be a more efficient alternative to LSTM. It is especially useful when computational resources are limited or when faster training is needed without a significant drop in performance.

For Simplicity and Baseline Comparison: Start with a basic RNN to establish a performance baseline. This approach helps to assess the improvement offered by more advanced architectures like LSTM or GRU.

Final Decision

LSTM is generally recommended for its effectiveness in managing long-term dependencies, which is crucial for comprehending context in complex text sequences. It tends to offer the best performance for tasks involving intricate textual data. Conversely, GRU provides a compelling alternative if you need a more efficient model with faster training times, offering performance similar to LSTM but with a simpler architecture.