```
import pandas as pd
import numpy as np
import re
import nltk
from nltk.corpus import stopwords
import plotly.express as px
import plotly.graph_objects as go
import tensorflow as tf
from google.colab import drive
from tensorflow.python.client import device_lib
from sklearn.metrics import classification_report

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
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, SimpleRNN, Dense, LSTM
```

```
print(device_lib.list_local_devices())

drive.mount('/content/drive')
nltk.download('stopwords')

if tf.config.list_physical_devices('GPU'):
    tf.config.experimental.set_memory_growth(tf.config.list_physical_devices('GPU')[0], True)
    tf.config.set_visible_devices(tf.config.list_physical_devices('GPU')[0], 'GPU')
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 18265766572125041073
xla_global_id: -1
]
Mounted at /content/drive
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

```
df = pd.read_csv('/content/drive/MyDrive/Datasets/Playstore_Reviews/reviews.csv')
```

```
df = df.head(5000)
```

```
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r"http\S+|www\S+|https\S+", "", text, flags=re.MULTILINE)
    text = re.sub(r"[^\w\s]", "", text)
    text = re.sub(r"\d+", "", text)
    stop_words = set(stopwords.words('english'))
    text = " ".join([word for word in text.split() if word not in stop_words])
    return text


def assign_sentiment_label(score):
    if score <= 2:
        return 'negative'
    elif score >= 4:
        return 'positive'
    else:
        return 'neutral'
```

```
X = df['content'].apply(preprocess_text)
y = df['score'].apply(assign_sentiment_label)
```

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

```
y = tf.keras.utils.to_categorical(y, 3)
```

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

```
X_validation, X_train = X_train[:300], X_train[300:]
y_validation, y_train = y_train[:300], y_train[300:]
```

```
X_train.shape, X_validation.shape, X_test.shape, y_train.shape, y_validation.shape, y_test.shape
```

```
((3700,), (300,), (1000,), (3700, 3), (300, 3), (1000, 3))
```

```
tokenizer = Tokenizer(num_words=1000)
tokenizer.fit_on_texts(X_train)

X_train = tokenizer.texts_to_sequences(X_train)
X_validation = tokenizer.texts_to_sequences(X_validation)
X_test = tokenizer.texts_to_sequences(X_test)

max_sequence_length = max([len(x) for x in X_train])
X_train = pad_sequences(X_train, maxlen=max_sequence_length)
X_validation = pad_sequences(X_validation, maxlen=max_sequence_length)
X_test = pad_sequences(X_test, maxlen=max_sequence_length)
```

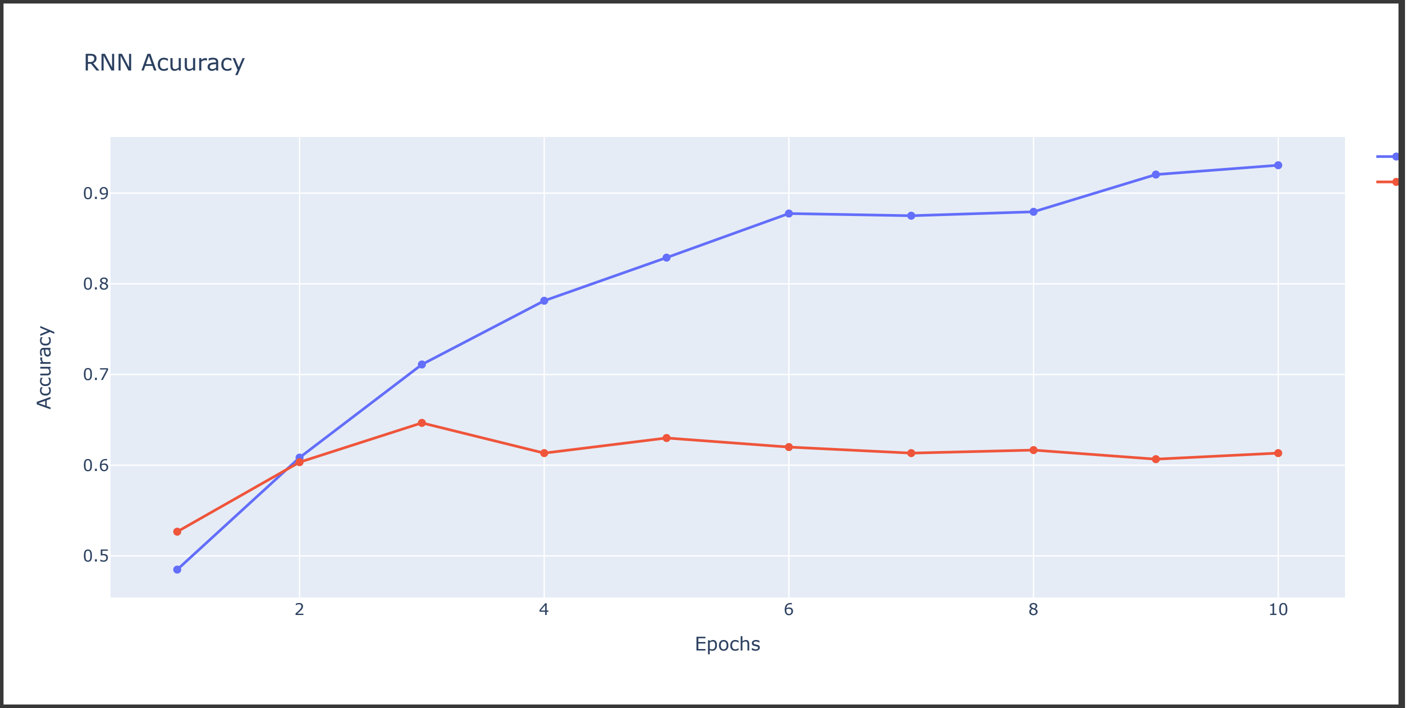
```
with tf.device('GPU'):
    model_rnn = Sequential()
    model_rnn.add(Embedding(1000, 32))
    model_rnn.add(SimpleRNN(32))
    model_rnn.add(Dense(3, activation='softmax'))
```

```
with tf.device('GPU'):
    epochs = 10
    model_rnn.compile(optimizer='adam',
                      loss= 'categorical_crossentropy',
                      metrics=['accuracy'])
    history_rnn = model_rnn.fit(X_train,
                                y_train,
                                epochs=epochs,
                                batch_size=64,
                                validation_data=(X_validation, y_validation))
```

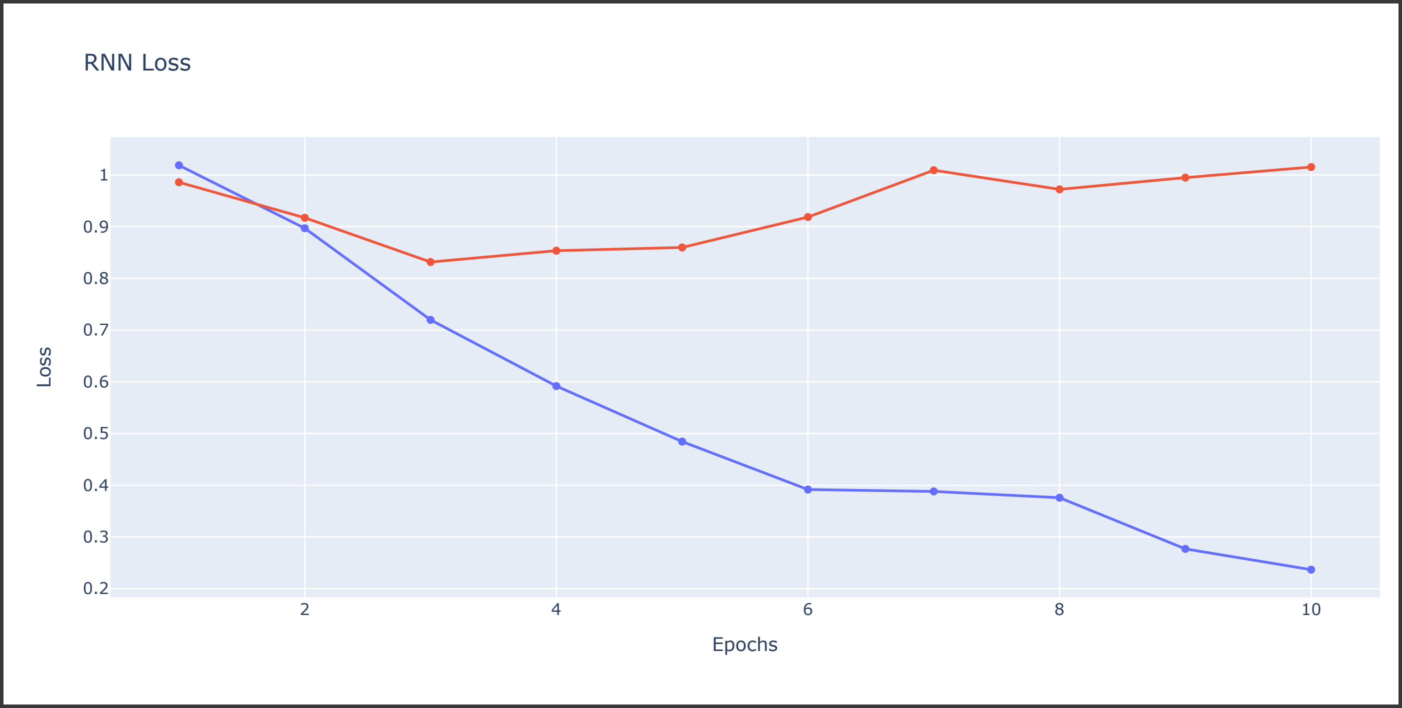
```
Epoch 1/10
58/58 [==============================] - 5s 66ms/step - loss: 1.0187 - accuracy: 0.4849 - val_loss: 0.9859 - val_accuracy: 0.
Epoch 2/10
58/58 [==============================] - 5s 91ms/step - loss: 0.8970 - accuracy: 0.6084 - val_loss: 0.9173 - val_accuracy: 0.
Epoch 3/10
58/58 [==============================] - 4s 66ms/step - loss: 0.7198 - accuracy: 0.7111 - val_loss: 0.8316 - val_accuracy: 0.
Epoch 4/10
58/58 [==============================] - 3s 59ms/step - loss: 0.5917 - accuracy: 0.7814 - val_loss: 0.8536 - val_accuracy: 0.
Epoch 5/10
58/58 [==============================] - 3s 59ms/step - loss: 0.4844 - accuracy: 0.8289 - val_loss: 0.8598 - val_accuracy: 0.
Epoch 6/10
58/58 [==============================] - 6s 102ms/step - loss: 0.3916 - accuracy: 0.8776 - val_loss: 0.9186 - val_accuracy: (
Epoch 7/10
58/58 [==============================] - 3s 59ms/step - loss: 0.3880 - accuracy: 0.8751 - val_loss: 1.0092 - val_accuracy: 0.
Epoch 8/10
58/58 [==============================] - 3s 60ms/step - loss: 0.3758 - accuracy: 0.8795 - val_loss: 0.9722 - val_accuracy: 0.
Epoch 9/10
58/58 [==============================] - 4s 70ms/step - loss: 0.2768 - accuracy: 0.9205 - val_loss: 0.9949 - val_accuracy: 0.
Epoch 10/10
58/58 [==============================] - 5s 87ms/step - loss: 0.2367 - accuracy: 0.9308 - val_loss: 1.0153 - val_accuracy: 0.
```

```
train_loss_rnn = history_rnn.history['loss']
validation_loss_rnn = history_rnn.history['val_loss']

train_acc_rnn = history_rnn.history['accuracy']
validation_acc_rnn = history_rnn.history['val_accuracy']
```

```
fig = go.Figure()

fig.add_trace(go.Scatter(x=list(range(1, epochs+1)), y=train_acc_rnn, mode='lines+markers', name='Train Accuracy'))
fig.add_trace(go.Scatter(x=list(range(1, epochs+1)), y=validation_acc_rnn, mode='lines+markers', name='Validation Accurac

fig.update_layout(title="RNN Acuuracy",
                  xaxis_title="Epochs",
                  yaxis_title="Accuracy")
fig.show()
```



```
fig = go.Figure()

fig.add_trace(go.Scatter(x=list(range(1, epochs+1)), y=train_loss_rnn, mode='lines+markers', name='Train Loss'))
fig.add_trace(go.Scatter(x=list(range(1, epochs+1)), y=validation_loss_rnn, mode='lines+markers', name='Validation Loss')

fig.update_layout(title="RNN Loss",
                  xaxis_title="Epochs",
                  yaxis_title="Loss")
fig.show()
```



```
test_predictions = model_rnn.predict(X_test)

predicted_labels = np.argmax(test_predictions, axis=1)
```

```
test_labels_1d = np.argmax(y_test, axis=1)
```

```
32/32 [==============================] - 1s 14ms/step
```

```
print(classification_report(test_labels_1d, predicted_labels))
```

```
              precision    recall  f1-score   support

           0       0.71      0.70      0.71       457
           1       0.20      0.15      0.17       144
           2       0.68      0.75      0.71       399

    accuracy                           0.64      1000
   macro avg       0.53      0.53      0.53      1000
weighted avg       0.62      0.64      0.63      1000
```

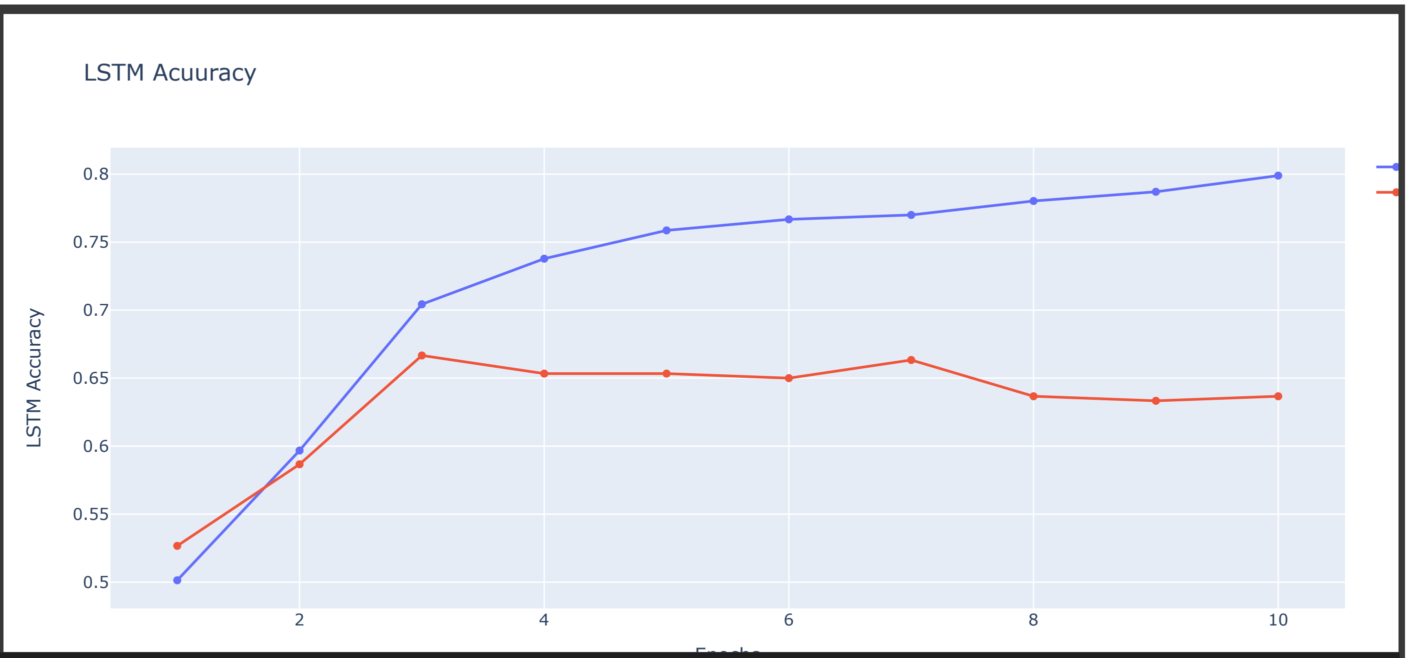
```
with tf.device('GPU'):
    model_lstm = Sequential()
    model_lstm.add(Embedding(1000, 32))
    model_lstm.add(LSTM(32))
    model_lstm.add(Dense(3, activation='softmax'))
```

```
with tf.device('GPU'):
    epochs = 10
    model_lstm.compile(optimizer='adam',
                       loss='binary_crossentropy',
                       metrics=['accuracy'])
    history_lstm = model_lstm.fit(X_train,
                                  y_train,
                                  epochs=epochs,
                                  batch_size=64,
                                  validation_data=(X_validation, y_validation))
```

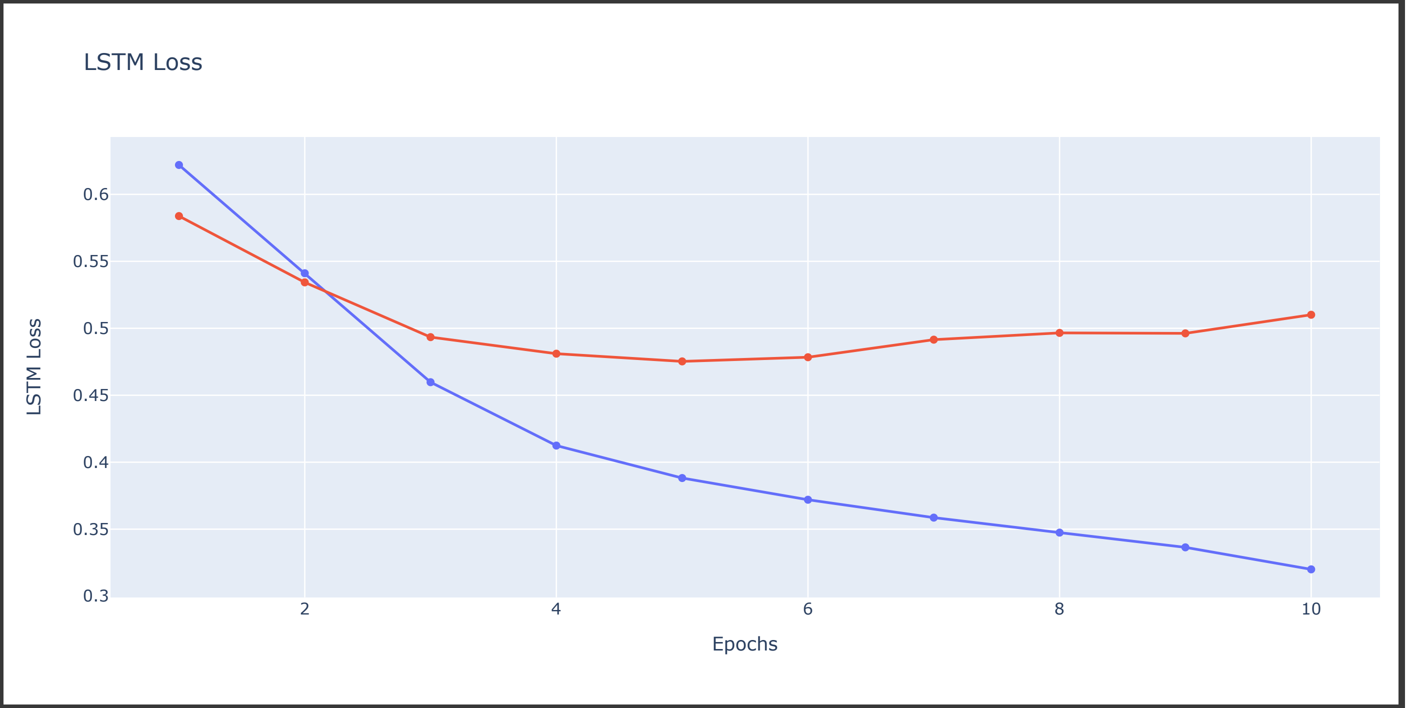
```
Epoch 1/10
58/58 [==============================] - 13s 178ms/step - loss: 0.6220 - accuracy: 0.5014 - val_loss: 0.5838 - val_accuracy:
Epoch 2/10
58/58 [==============================] - 8s 132ms/step - loss: 0.5411 - accuracy: 0.5968 - val_loss: 0.5343 - val_accuracy: (
Epoch 3/10
58/58 [==============================] - 9s 150ms/step - loss: 0.4598 - accuracy: 0.7043 - val_loss: 0.4934 - val_accuracy: (
Epoch 4/10
58/58 [==============================] - 9s 156ms/step - loss: 0.4124 - accuracy: 0.7378 - val_loss: 0.4810 - val_accuracy: (
Epoch 5/10
58/58 [==============================] - 7s 123ms/step - loss: 0.3882 - accuracy: 0.7586 - val_loss: 0.4753 - val_accuracy: (
Epoch 6/10
58/58 [==============================] - 9s 159ms/step - loss: 0.3720 - accuracy: 0.7668 - val_loss: 0.4784 - val_accuracy: (
Epoch 7/10
58/58 [==============================] - 7s 127ms/step - loss: 0.3586 - accuracy: 0.7700 - val_loss: 0.4915 - val_accuracy: (
Epoch 8/10
58/58 [==============================] - 9s 157ms/step - loss: 0.3474 - accuracy: 0.7803 - val_loss: 0.4966 - val_accuracy: (
Epoch 9/10
58/58 [==============================] - 7s 122ms/step - loss: 0.3364 - accuracy: 0.7870 - val_loss: 0.4962 - val_accuracy: (
Epoch 10/10
58/58 [==============================] - 9s 155ms/step - loss: 0.3200 - accuracy: 0.7989 - val_loss: 0.5101 - val_accuracy: (
```

```
train_loss_lstm = history_lstm.history['loss']
validation_loss_lstm = history_lstm.history['val_loss']

train_acc_lstm = history_lstm.history['accuracy']
validation_acc_lstm = history_lstm.history['val_accuracy']
```

```
fig = go.Figure()

fig.add_trace(go.Scatter(x=list(range(1, epochs+1)), y=train_acc_lstm, mode='lines+markers', name='Train Accuracy'))
fig.add_trace(go.Scatter(x=list(range(1, epochs+1)), y=validation_acc_lstm, mode='lines+markers', name='Validation Accura

fig.update_layout(title="LSTM Acuuracy",
                  xaxis_title="Epochs",
                  yaxis_title="LSTM Accuracy")
fig.show()
```



```
fig = go.Figure()

fig.add_trace(go.Scatter(x=list(range(1, epochs+1)), y=train_loss_lstm, mode='lines+markers', name='Train Loss'))
fig.add_trace(go.Scatter(x=list(range(1, epochs+1)), y=validation_loss_lstm, mode='lines+markers', name='Validation Loss'

fig.update_layout(title="LSTM Loss",
                  xaxis_title="Epochs",
                  yaxis_title="LSTM Loss")
fig.show()
```



```
test_predictions = model_lstm.predict(X_test)

predicted_labels = np.argmax(test_predictions, axis=1)

test_labels_1d = np.argmax(y_test, axis=1)
```

```
    32/32 [==============================] - 1s 24ms/step
```

```
print(classification_report(test_labels_1d, predicted_labels))
```

```
              precision    recall  f1-score   support

           0       0.69      0.78      0.73       457
           1       0.24      0.04      0.07       144
           2       0.68      0.77      0.72       399

    accuracy                           0.67      1000
   macro avg       0.54      0.53      0.51      1000
weighted avg       0.62      0.67      0.63      1000
```