```
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: 5261938452248922550
xla_global_id: -1
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 14328594432
locality {
  bus_id: 1
  links {
  }
}
incarnation: 3532726050434287717
physical_device_desc: "device: 0, name: Tesla T4, pci bus id: 0000:00:04.0, compute capability: 7.5"
xla_global_id: 416903419
]
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/IMDB_Movie_Review/Reviews.csv')
```

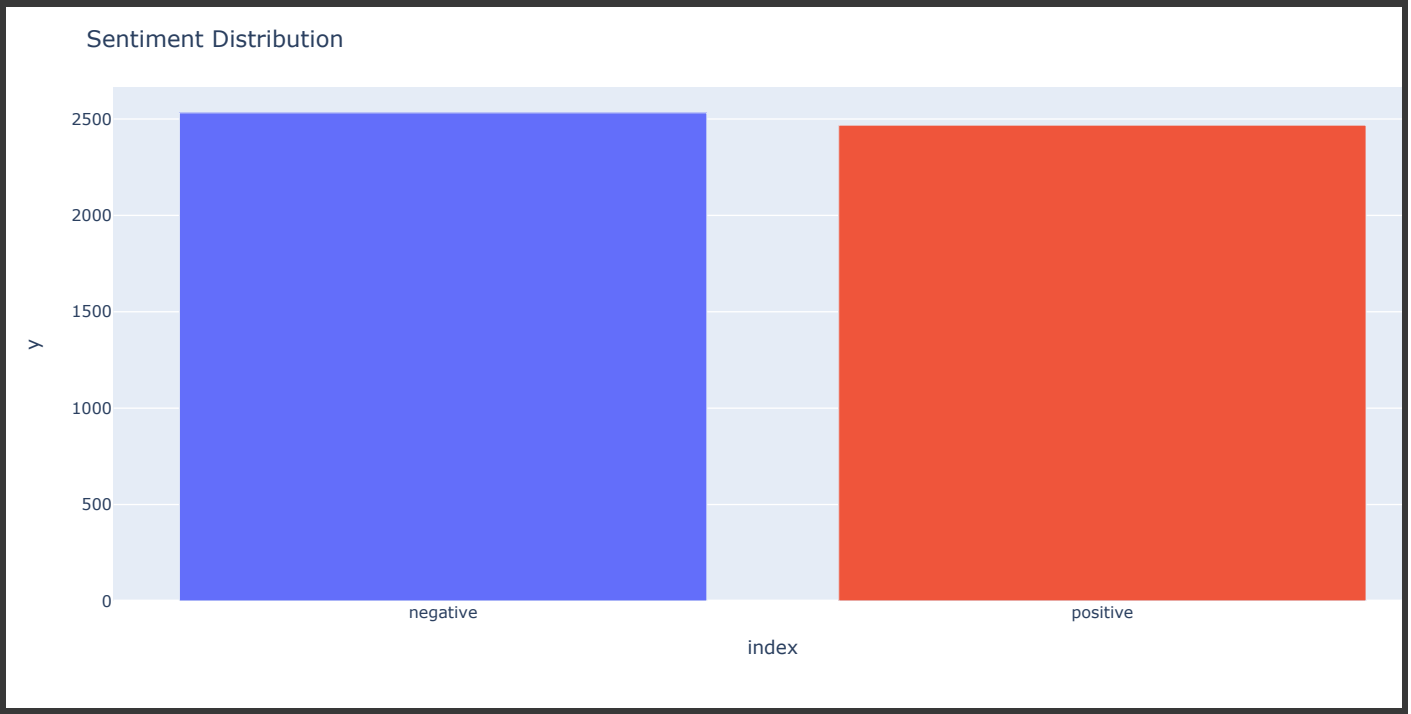
```
df = df.head(5000)
```

```
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
```

```
num_labels = 2

X = df['review'].apply(preprocess_text)
y = df['sentiment']
```

```
sentiment_counts = y.value_counts()
fig = px.bar(sentiment_counts, x=sentiment_counts.index, y=sentiment_counts.values, color=sentiment_counts.index)
```

```
fig.update_layout(title="Sentiment Distribution")
fig.show()
```




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

```
y = tf.keras.utils.to_categorical(y, 2)
```

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

```
validation_num = 500

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

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

```
((3500,), (500,), (1000,), (3500, 2), (500, 2), (1000, 2))
```

```
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(2, activation='softmax'))
```

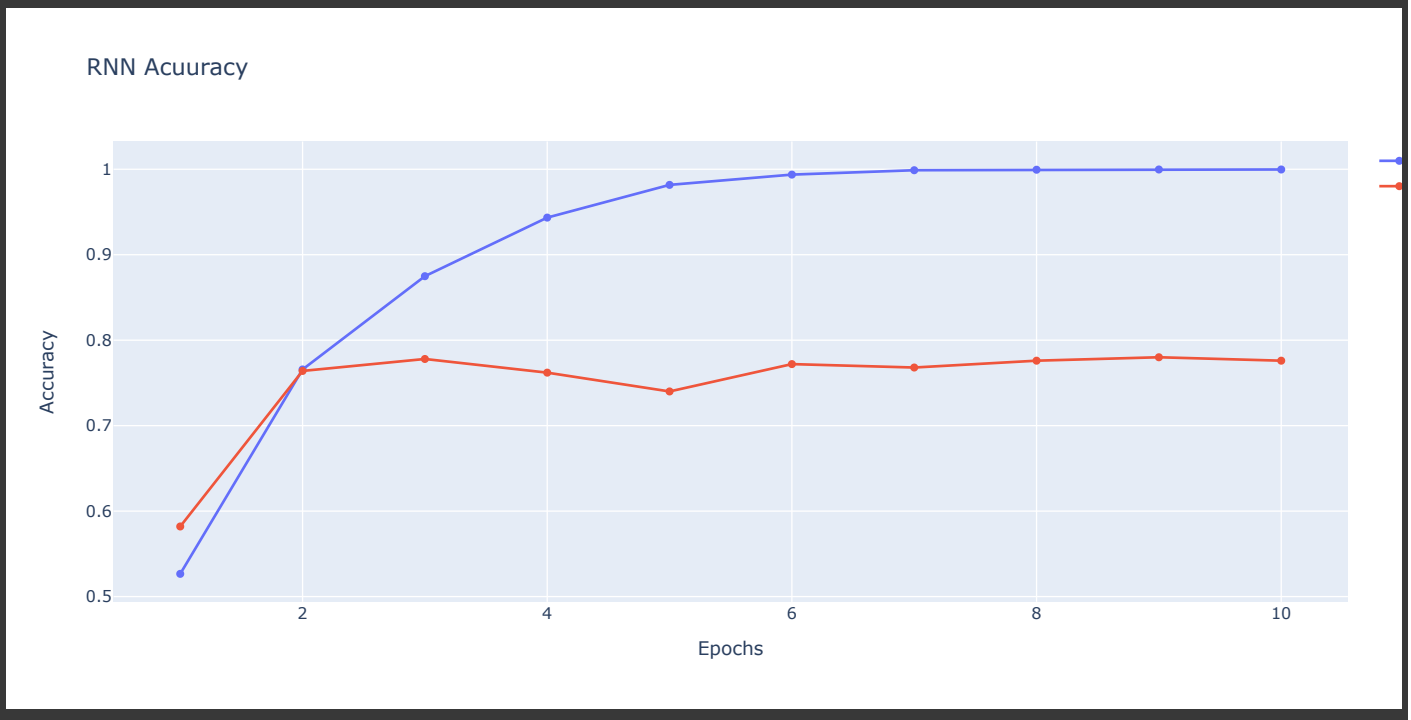
```
with tf.device('GPU'):
    epochs = 10
    model_rnn.compile(optimizer='adam',
                      loss= 'categorical_crossentropy',
```

```
                    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
55/55 [==============================] - 45s 713ms/step - loss: 0.6943 - accuracy: 0.5266 - val_loss: 0.6757 - val_accuracy:
Epoch 2/10
55/55 [==============================] - 32s 579ms/step - loss: 0.5402 - accuracy: 0.7657 - val_loss: 0.5037 - val_accuracy:
Epoch 3/10
55/55 [==============================] - 31s 573ms/step - loss: 0.3239 - accuracy: 0.8749 - val_loss: 0.4869 - val_accuracy:
Epoch 4/10
55/55 [==============================] - 28s 508ms/step - loss: 0.1824 - accuracy: 0.9434 - val_loss: 0.5177 - val_accuracy:
Epoch 5/10
55/55 [==============================] - 28s 513ms/step - loss: 0.0926 - accuracy: 0.9817 - val_loss: 0.6344 - val_accuracy:
Epoch 6/10
55/55 [==============================] - 28s 512ms/step - loss: 0.0451 - accuracy: 0.9937 - val_loss: 0.6166 - val_accuracy:
Epoch 7/10
55/55 [==============================] - 29s 529ms/step - loss: 0.0219 - accuracy: 0.9989 - val_loss: 0.6669 - val_accuracy:
Epoch 8/10
55/55 [==============================] - 29s 532ms/step - loss: 0.0121 - accuracy: 0.9994 - val_loss: 0.7040 - val_accuracy:
Epoch 9/10
55/55 [==============================] - 28s 503ms/step - loss: 0.0082 - accuracy: 0.9997 - val_loss: 0.7386 - val_accuracy:
Epoch 10/10
55/55 [==============================] - 36s 651ms/step - loss: 0.0080 - accuracy: 0.9997 - val_loss: 0.7713 - val_accuracy:
```

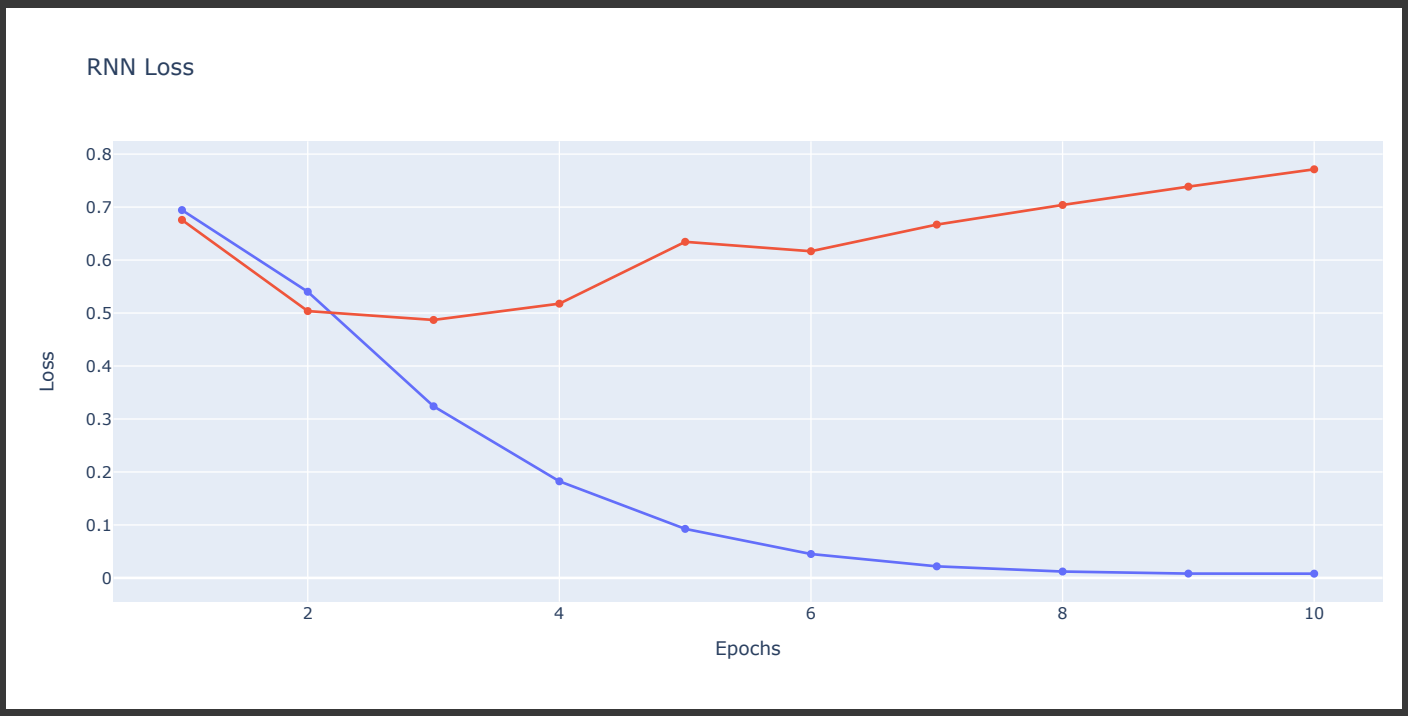
```
train_loss_rnn = history_rnn.history['loss']
validation_loss_rnn = history_rnn.history['val_loss']

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 [==============================] - 3s 78ms/step
```

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

```
              precision    recall  f1-score   support

           0       0.82      0.71      0.76       530
           1       0.71      0.82      0.76       470

    accuracy                           0.76      1000
   macro avg       0.76      0.76      0.76      1000
weighted avg       0.77      0.76      0.76      1000
```

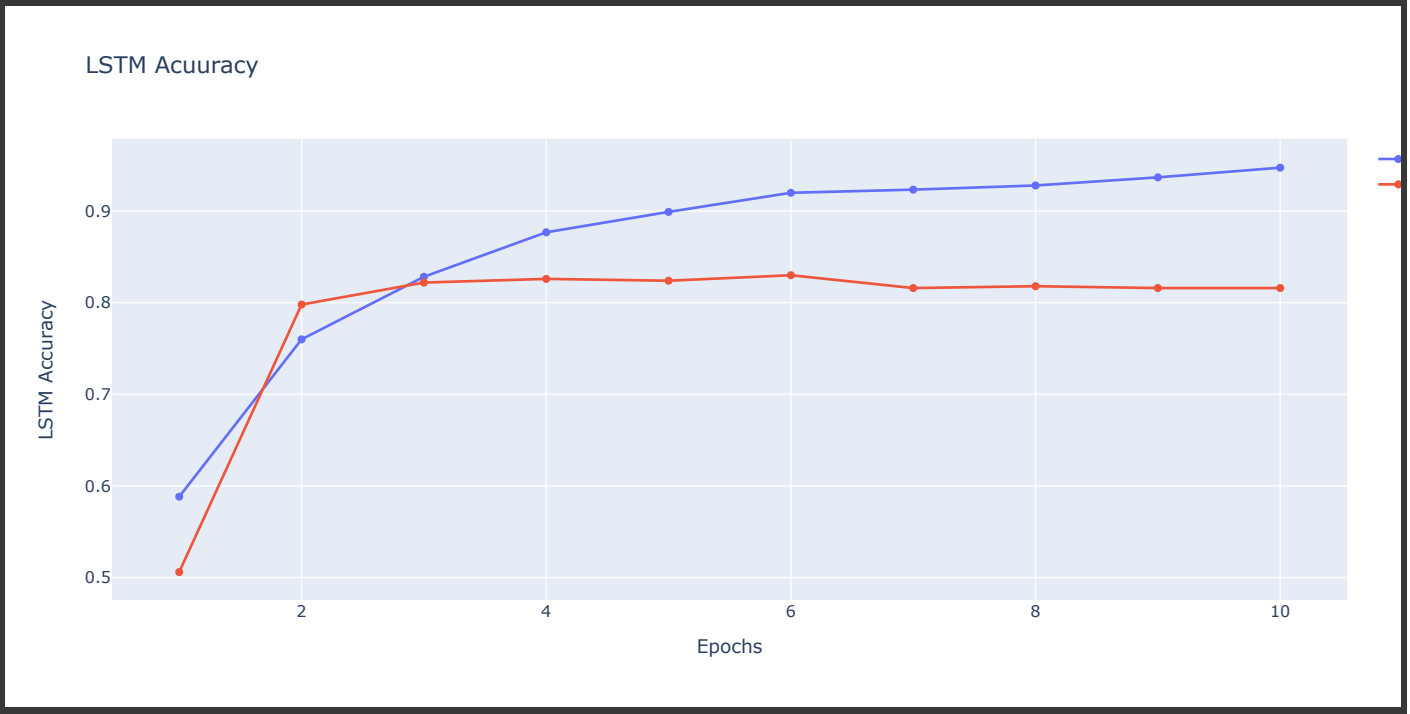
```
with tf.device('GPU'):
    model_lstm = Sequential()
    model_lstm.add(Embedding(1000, 32))
    model_lstm.add(LSTM(32))
    model_lstm.add(Dense(2, activation='softmax'))
```

```
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))
```

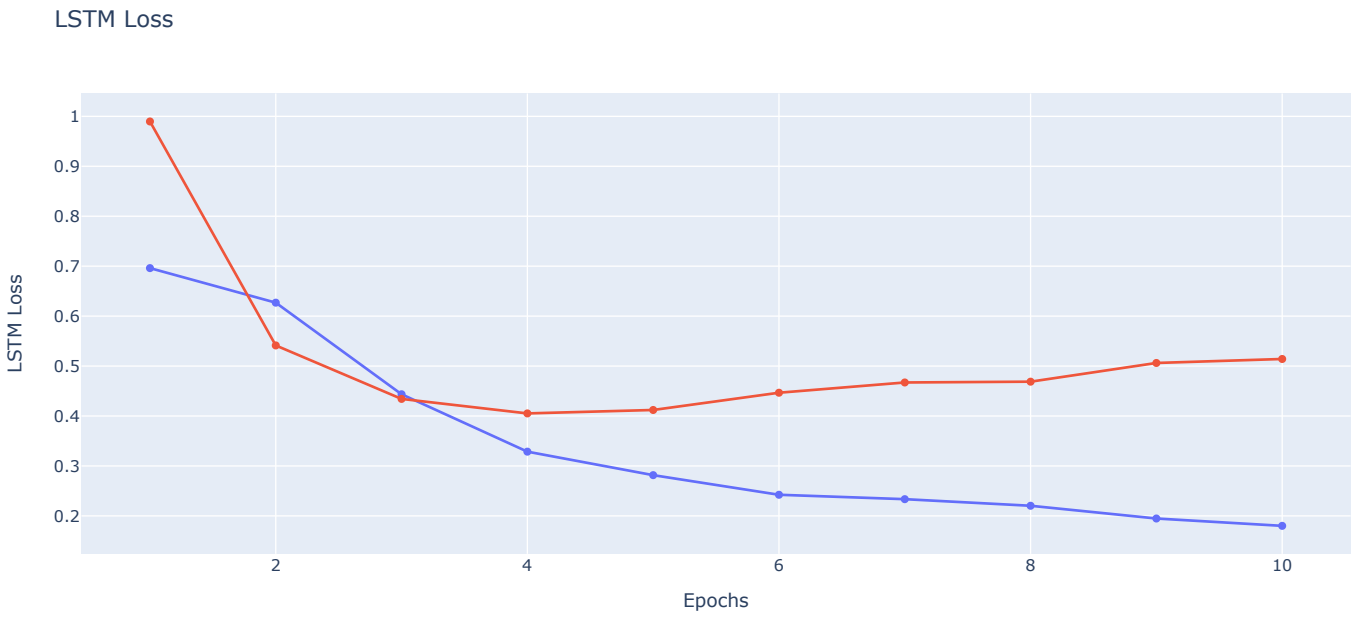
```
Epoch 1/10
55/55 [==============================] - 18s 235ms/step - loss: 0.6961 - accuracy: 0.5883 - val_loss: 0.9897 - val_accuracy:
Epoch 2/10
55/55 [==============================] - 5s 86ms/step - loss: 0.6269 - accuracy: 0.7600 - val_loss: 0.5413 - val_accuracy: 0.
Epoch 3/10
55/55 [==============================] - 4s 75ms/step - loss: 0.4435 - accuracy: 0.8283 - val_loss: 0.4342 - val_accuracy: 0.
Epoch 4/10
55/55 [==============================] - 2s 45ms/step - loss: 0.3286 - accuracy: 0.8769 - val_loss: 0.4052 - val_accuracy: 0.
Epoch 5/10
55/55 [==============================] - 3s 51ms/step - loss: 0.2816 - accuracy: 0.8991 - val_loss: 0.4121 - val_accuracy: 0.
Epoch 6/10
55/55 [==============================] - 2s 28ms/step - loss: 0.2423 - accuracy: 0.9200 - val_loss: 0.4466 - val_accuracy: 0.
Epoch 7/10
55/55 [==============================] - 1s 27ms/step - loss: 0.2335 - accuracy: 0.9234 - val_loss: 0.4671 - val_accuracy: 0.
Epoch 8/10
55/55 [==============================] - 2s 38ms/step - loss: 0.2203 - accuracy: 0.9280 - val_loss: 0.4687 - val_accuracy: 0.
Epoch 9/10
55/55 [==============================] - 2s 39ms/step - loss: 0.1948 - accuracy: 0.9369 - val_loss: 0.5062 - val_accuracy: 0.
Epoch 10/10
55/55 [==============================] - 1s 19ms/step - loss: 0.1801 - accuracy: 0.9474 - val_loss: 0.5142 - val_accuracy: 0.
```

```
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 9ms/step
```

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

```
              precision    recall  f1-score   support

           0       0.88      0.78      0.83       530
           1       0.78      0.88      0.83       470

    accuracy                           0.83      1000
   macro avg       0.83      0.83      0.83      1000
weighted avg       0.83      0.83      0.83      1000
```