```
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

```
import pandas as pd
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import re
from nltk.stem import WordNetLemmatizer
from tensorflow.keras.preprocessing.text import Tokenizer
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import CountVectorizer
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import classification_report
```

```
data = pd.read_csv("/content/drive/MyDrive/TheSocialDilemma.csv")
```

```
data.head()
```

	user_name	user_location	user_description	user_created	user_followers	user_friends	user_favourites	user_verified	date	text	hashtags	source	is_retweet	Sentiment
0	Mari Smith	San Diego, California	Premier Facebook Marketing Expert I Social Med...	2007-09-11 22:22:51	579942	288625	11610	False	2020-09-16 20:55:33	@musicmadmarc @SocialDilemma_ @netflix @Facebo...	NaN	Twitter Web App	False	Neutral
1	Mari Smith	San Diego, California	Premier Facebook Marketing Expert I Social Med...	2007-09-11 22:22:51	579942	288625	11610	False	2020-09-16 20:53:17	@musicmadmarc @SocialDilemma_ @netflix @Facebo...	NaN	Twitter Web App	False	Neutral
2	Varun Tyagi	Goa, India	Indian I Tech Solution Artist & Hospitality Ex...	2009-09-06 10:36:01	257	204	475	False	2020-09-16 20:51:57	Go watch "The Social Dilemma" on Netflix!\n\nI...	NaN	Twitter for iPhone	False	Positive
3	Casey Conway	Sydney, New South Wales	Head of Diversity & Inclusion @RugbyAU I It's ...	2012-12-28 21:45:06	11782	1033	12219	True	2020-09-16 20:51:46	I watched #TheSocialDilemma last night. I'm sc...	['TheSocialDilemma']	Twitter for iPhone	False	Negative
4	Charlotte Paul	Darlington	Instagram Charlottejyates	2012-05-28 20:43:08	278	387	5850	False	2020-09-16 20:51:11	The problem of me being on my phone most the t...	['TheSocialDilemma']	Twitter for iPhone	False	Positive

Next steps: Generate code with data View recommended plots

```
print(data.columns)
```

```
Index(['user_name', 'user_location', 'user_description', 'user_created',
       'user_followers', 'user_friends', 'user_favourites', 'user_verified',
       'date', 'text', 'hashtags', 'source', 'is_retweet', 'Sentiment'],
      dtype='object')
```

```
data = data[['text', 'Sentiment']]
```

```
data.head()
```

	text	Sentiment
0	@musicmadmarc @SocialDilemma_ @netflix @Facebo...	Neutral
1	@musicmadmarc @SocialDilemma_ @netflix @Facebo...	Neutral
2	Go watch “The Social Dilemma” on Netflix!\n\nI...	Positive
3	I watched #TheSocialDilemma last night. I’m sc...	Negative
4	The problem of me being on my phone most the t...	Positive

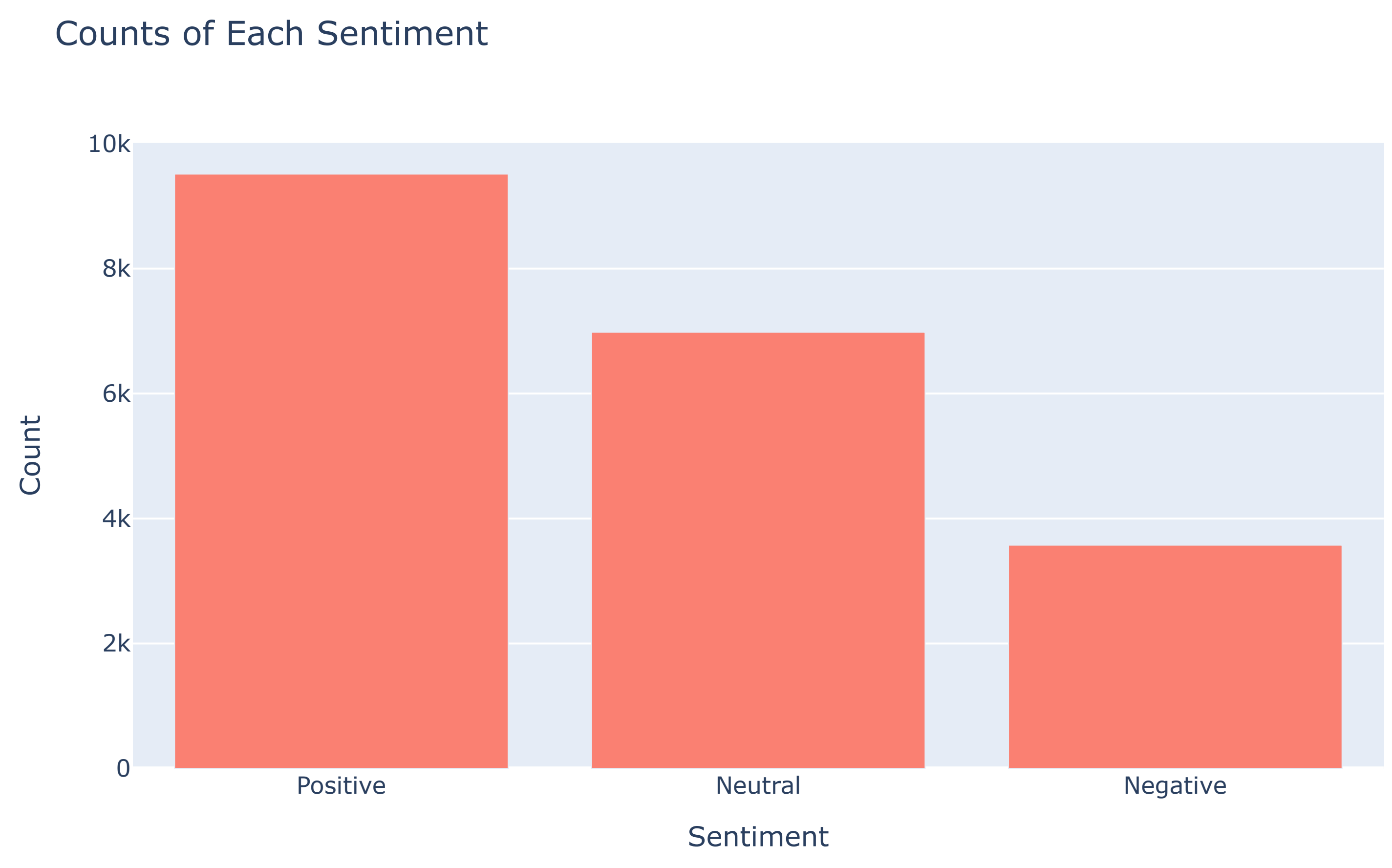
Next steps: Generate code with data View recommended plots

```
import plotly.graph_objects as go

# Assuming data contains 'text' and 'Sentiment' columns
bar_fig = go.Figure([go.Bar(x=data['Sentiment'].value_counts().index,
                             y=data['Sentiment'].value_counts().tolist(),
                             marker_color='salmon')])  # Changing color to salmon

# Update layout of the figure
bar_fig.update_layout(
    title="Counts of Each Sentiment",
    xaxis_title="Sentiment",
    yaxis_title="Count",
    width=800,
    height=500
)

# Show the figure
bar_fig.show()
```




```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
import tensorflow as tf
import re
```

```
def clean_text(text):
    text = text.lower()
    text = re.sub('\[.*?\]', '', text)
    text = re.sub('https?://\S+|www\.\S+', '', text)
    text = re.sub('\n', '', text)
    text = " ".join(filter(lambda x:x[0]!="@", text.split()))
    return text

data['text'] = data['text'].apply(lambda x: clean_text(x))

X = data['text']
y = data['Sentiment'].map({'Negative':0, 'Neutral':1, 'Positive':2})

train_size = int(len(data)*0.8)
X_train, y_train = X[:train_size], y[:train_size]
X_test, y_test = X[train_size:], y[train_size:]
```

```
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_train shape: (16054,)
y_train shape: (16054,)
X_test shape: (4014,)
y_test shape: (4014,)
```

```
vocab_size = 10000
embedding_dim = 16
max_length = 90
tokenizer = Tokenizer(num_words=vocab_size)
tokenizer.fit_on_texts(X_train)
word_index = tokenizer.word_index

train_sequences = tokenizer.texts_to_sequences(X_train)
train_padded = pad_sequences(train_sequences, maxlen=max_length, padding='pre', truncating='pre')
test_sequences = tokenizer.texts_to_sequences(X_test)
test_padded = pad_sequences(test_sequences, maxlen=max_length, padding='pre', truncating='pre')

print("Shape of train_padded:", train_padded.shape)
print("Shape of test_padded:", test_padded.shape)
```

```
Shape of train_padded: (16054, 90)
Shape of test_padded: (4014, 90)
```

```
from keras.utils import to_categorical
y_train_encoded = to_categorical(y_train)
y_test_encoded = to_categorical(y_test)

# Define the model
model_secq = tf.keras.Sequential([
    tf.keras.layers.Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=max_length),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')  # Adjusted output units to match the number of classes
])
```

```
model_secq.summary()
```

```
Model: "sequential_18"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_18 (Embedding)    (None, 90, 16)            160000

 global_average_pooling1d_1  (None, 16)                0
 0 (GlobalAveragePooling1D)

 dense_29 (Dense)            (None, 32)                544

 dense_30 (Dense)            (None, 3)                 99

=================================================================
Total params: 160643 (627.51 KB)
Trainable params: 160643 (627.51 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

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

# Fit the model with the padded sequences and one-hot encoded labels
history = model_secq.fit(train_padded, y_train_encoded,
                     epochs=20,
                     batch_size=512,
                     validation_data=(test_padded, y_test_encoded))
```

```
Epoch 1/20
32/32 [==============================] - 2s 27ms/step - loss: 1.0740 - accuracy: 0.4749 - val_loss: 1.0534 - val_accuracy: 0.4706
Epoch 2/20
32/32 [==============================] - 1s 16ms/step - loss: 1.0352 - accuracy: 0.4749 - val_loss: 1.0291 - val_accuracy: 0.4706
Epoch 3/20
32/32 [==============================] - 1s 17ms/step - loss: 1.0221 - accuracy: 0.4749 - val_loss: 1.0260 - val_accuracy: 0.4706
Epoch 4/20
32/32 [==============================] - 1s 16ms/step - loss: 1.0177 - accuracy: 0.4749 - val_loss: 1.0217 - val_accuracy: 0.4706
Epoch 5/20
32/32 [==============================] - 1s 17ms/step - loss: 1.0111 - accuracy: 0.4749 - val_loss: 1.0147 - val_accuracy: 0.4706
Epoch 6/20
32/32 [==============================] - 1s 16ms/step - loss: 1.0003 - accuracy: 0.4793 - val_loss: 1.0027 - val_accuracy: 0.4910
Epoch 7/20
32/32 [==============================] - 1s 17ms/step - loss: 0.9818 - accuracy: 0.5040 - val_loss: 0.9832 - val_accuracy: 0.5399
Epoch 8/20
32/32 [==============================] - 0s 10ms/step - loss: 0.9540 - accuracy: 0.5996 - val_loss: 0.9553 - val_accuracy: 0.6091
Epoch 9/20
32/32 [==============================] - 0s 10ms/step - loss: 0.9167 - accuracy: 0.6342 - val_loss: 0.9212 - val_accuracy: 0.6286
Epoch 10/20
32/32 [==============================] - 0s 12ms/step - loss: 0.8746 - accuracy: 0.6510 - val_loss: 0.8866 - val_accuracy: 0.6343
Epoch 11/20
32/32 [==============================] - 0s 11ms/step - loss: 0.8326 - accuracy: 0.6616 - val_loss: 0.8538 - val_accuracy: 0.6450
Epoch 12/20
32/32 [==============================] - 0s 12ms/step - loss: 0.7915 - accuracy: 0.6788 - val_loss: 0.8227 - val_accuracy: 0.6622
Epoch 13/20
32/32 [==============================] - 0s 11ms/step - loss: 0.7517 - accuracy: 0.7042 - val_loss: 0.7930 - val_accuracy: 0.6719
Epoch 14/20
32/32 [==============================] - 0s 12ms/step - loss: 0.7119 - accuracy: 0.7260 - val_loss: 0.7653 - val_accuracy: 0.6826
Epoch 15/20
32/32 [==============================] - 0s 12ms/step - loss: 0.6710 - accuracy: 0.7457 - val_loss: 0.7327 - val_accuracy: 0.6991
Epoch 16/20
32/32 [==============================] - 0s 10ms/step - loss: 0.6316 - accuracy: 0.7636 - val_loss: 0.7035 - val_accuracy: 0.7133
Epoch 17/20
32/32 [==============================] - 0s 11ms/step - loss: 0.5936 - accuracy: 0.7825 - val_loss: 0.6801 - val_accuracy: 0.7299
Epoch 18/20
32/32 [==============================] - 0s 12ms/step - loss: 0.5566 - accuracy: 0.8018 - val_loss: 0.6515 - val_accuracy: 0.7441
Epoch 19/20
32/32 [==============================] - 0s 11ms/step - loss: 0.5217 - accuracy: 0.8182 - val_loss: 0.6268 - val_accuracy: 0.7591
Epoch 20/20
32/32 [==============================] - 0s 11ms/step - loss: 0.4884 - accuracy: 0.8326 - val_loss: 0.6049 - val_accuracy: 0.7673
```

```
from sklearn.metrics import classification_report
import numpy as np

# Preprocess the test data
X_test_cleaned = X_test.apply(clean_text)

# Tokenize and pad the test data
test_sequences = tokenizer.texts_to_sequences(X_test_cleaned)
test_padded = pad_sequences(test_sequences, maxlen=max_length, padding='pre', truncating='pre')

# Predict sentiment labels for test data
pred = model_secq.predict(test_padded)

# Convert predicted labels to numerical form
predicted_labels = np.argmax(pred, axis=1)

# Generate classification report
print(classification_report(y_test, predicted_labels))
```

```
126/126 [==============================] - 1s 4ms/step
              precision    recall  f1-score   support

           0       0.72      0.30      0.42       730
           1       0.75      0.85      0.79      1395
           2       0.79      0.89      0.84      1889

    accuracy                           0.77      4014
   macro avg       0.75      0.68      0.68      4014
weighted avg       0.76      0.77      0.75      4014
```

```
from tensorflow.keras import layers, models
from keras import models
from keras import layers

# Define the model
model_Rnn = models.Sequential()
model_Rnn.add(layers.Embedding(input_dim=vocab_size, output_dim=16, input_length=max_length))
model_Rnn.add(layers.SimpleRNN(48))
model_Rnn.add(layers.Dense(3, activation='softmax'))

# Compile the model
model_Rnn.compile(optimizer='adam',
               loss='categorical_crossentropy',
               metrics=['accuracy'])

# Print the summary of the model
model_Rnn.summary()
```

```
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_14 (Embedding)    (None, 90, 16)            160000

 simple_rnn_2 (SimpleRNN)    (None, 48)                3120

 dense_24 (Dense)            (None, 3)                 147

=================================================================
Total params: 163267 (637.76 KB)
Trainable params: 163267 (637.76 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Fit the model
history = model_Rnn.fit(train_padded,
                     y_train_encoded,
                     epochs=19,
                     batch_size=512,
                     validation_split=0.2)
```

```
Epoch 1/19
26/26 [==============================] - 3s 133ms/step - loss: 0.0041 - accuracy: 0.9992 - val_loss: 0.8000 - val_accuracy: 0.8449
Epoch 2/19
26/26 [==============================] - 3s 106ms/step - loss: 0.0039 - accuracy: 0.9995 - val_loss: 0.7963 - val_accuracy: 0.8452
Epoch 3/19
26/26 [==============================] - 3s 112ms/step - loss: 0.0036 - accuracy: 0.9993 - val_loss: 0.8148 - val_accuracy: 0.8434
Epoch 4/19
26/26 [==============================] - 2s 93ms/step - loss: 0.0036 - accuracy: 0.9994 - val_loss: 0.8200 - val_accuracy: 0.8437
Epoch 5/19
26/26 [==============================] - 3s 104ms/step - loss: 0.0034 - accuracy: 0.9993 - val_loss: 0.8315 - val_accuracy: 0.8446
Epoch 6/19
26/26 [==============================] - 2s 74ms/step - loss: 0.0033 - accuracy: 0.9995 - val_loss: 0.8299 - val_accuracy: 0.8446
Epoch 7/19
26/26 [==============================] - 2s 74ms/step - loss: 0.0030 - accuracy: 0.9995 - val_loss: 0.8299 - val_accuracy: 0.8452
Epoch 8/19
26/26 [==============================] - 2s 68ms/step - loss: 0.0032 - accuracy: 0.9995 - val_loss: 0.8353 - val_accuracy: 0.8427
Epoch 9/19
26/26 [==============================] - 2s 68ms/step - loss: 0.0032 - accuracy: 0.9996 - val_loss: 0.8673 - val_accuracy: 0.8396
Epoch 10/19
26/26 [==============================] - 2s 69ms/step - loss: 0.0032 - accuracy: 0.9993 - val_loss: 0.8578 - val_accuracy: 0.8434
Epoch 11/19
26/26 [==============================] - 2s 81ms/step - loss: 0.0036 - accuracy: 0.9991 - val_loss: 0.8483 - val_accuracy: 0.8424
Epoch 12/19
26/26 [==============================] - 3s 105ms/step - loss: 0.0039 - accuracy: 0.9992 - val_loss: 0.8548 - val_accuracy: 0.8446
Epoch 13/19
26/26 [==============================] - 2s 72ms/step - loss: 0.0040 - accuracy: 0.9993 - val_loss: 0.8802 - val_accuracy: 0.8415
Epoch 14/19
26/26 [==============================] - 2s 68ms/step - loss: 0.0040 - accuracy: 0.9992 - val_loss: 0.8792 - val_accuracy: 0.8356
Epoch 15/19
26/26 [==============================] - 2s 69ms/step - loss: 0.0031 - accuracy: 0.9995 - val_loss: 0.8814 - val_accuracy: 0.8399
Epoch 16/19
26/26 [==============================] - 2s 69ms/step - loss: 0.0030 - accuracy: 0.9993 - val_loss: 0.8676 - val_accuracy: 0.8415
Epoch 17/19
26/26 [==============================] - 2s 75ms/step - loss: 0.0029 - accuracy: 0.9995 - val_loss: 0.8622 - val_accuracy: 0.8412
Epoch 18/19
26/26 [==============================] - 2s 79ms/step - loss: 0.0028 - accuracy: 0.9993 - val_loss: 0.8813 - val_accuracy: 0.8384
Epoch 19/19
26/26 [==============================] - 3s 102ms/step - loss: 0.0030 - accuracy: 0.9993 - val_loss: 0.8756 - val_accuracy: 0.8405
```

```
# Preprocess the test data
X_test_cleaned = X_test.apply(clean_text)

# Tokenize and pad the test data
test_sequences = tokenizer.texts_to_sequences(X_test_cleaned)
test_padded = pad_sequences(test_sequences, maxlen=max_length, padding='pre', truncating='pre')

# Predict sentiment labels for test data using model_Rnn
pred = model_Rnn.predict(test_padded)

# Convert predicted probabilities to predicted labels
predicted_labels = np.argmax(pred, axis=1)

# Generate classification report
print(classification_report(y_test, predicted_labels))
```

```
126/126 [==============================] - 1s 6ms/step
              precision    recall  f1-score   support

           0       0.78      0.67      0.72       730
           1       0.82      0.90      0.86      1395
           2       0.87      0.86      0.86      1889

    accuracy                           0.84      4014
   macro avg       0.82      0.81      0.82      4014
weighted avg       0.84      0.84      0.84      4014
```

```
# Define the model
model_LSTM = models.Sequential()
model_LSTM.add(layers.Embedding(input_dim=vocab_size, output_dim=16, input_length=max_length))
model_LSTM.add(layers.LSTM(48))
model_LSTM.add(layers.Dense(3, activation='softmax'))

# Print the summary of the model
model_LSTM.summary()

# Compile the model
model_LSTM.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_15 (Embedding)    (None, 90, 16)            160000

 lstm_2 (LSTM)               (None, 48)                12480

 dense_25 (Dense)            (None, 3)                 147

=================================================================
Total params: 172627 (674.32 KB)
Trainable params: 172627 (674.32 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Fit the model
history = model_LSTM.fit(train_padded,
                    y_train_encoded,
                    epochs=20,
                    batch_size=512,
                    validation_split=0.2)
```

```
Epoch 1/20
26/26 [==============================] - 16s 410ms/step - loss: 1.0487 - accuracy: 0.4889 - val_loss: 0.9956 - val_accuracy: 0.5665
Epoch 2/20
26/26 [==============================] - 8s 304ms/step - loss: 0.9585 - accuracy: 0.5682 - val_loss: 0.9153 - val_accuracy: 0.5867
Epoch 3/20
26/26 [==============================] - 6s 245ms/step - loss: 0.8628 - accuracy: 0.6070 - val_loss: 0.8022 - val_accuracy: 0.6450
Epoch 4/20
26/26 [==============================] - 8s 307ms/step - loss: 0.7342 - accuracy: 0.6803 - val_loss: 0.7229 - val_accuracy: 0.7023
Epoch 5/20
26/26 [==============================] - 6s 240ms/step - loss: 0.5900 - accuracy: 0.7642 - val_loss: 0.6038 - val_accuracy: 0.7565
Epoch 6/20
26/26 [==============================] - 8s 309ms/step - loss: 0.5017 - accuracy: 0.7944 - val_loss: 0.5522 - val_accuracy: 0.7761
Epoch 7/20
```

```
26/26 [==============================] - 6s 242ms/step - loss: 0.3976 - accuracy: 0.8369 - val_loss: 0.5063 - val_accuracy: 0.8050
Epoch 8/20
26/26 [==============================] - 9s 342ms/step - loss: 0.3160 - accuracy: 0.8876 - val_loss: 0.4752 - val_accuracy: 0.8259
Epoch 9/20
26/26 [==============================] - 6s 241ms/step - loss: 0.2393 - accuracy: 0.9238 - val_loss: 0.5039 - val_accuracy: 0.8387
Epoch 10/20
26/26 [==============================] - 8s 306ms/step - loss: 0.1955 - accuracy: 0.9424 - val_loss: 0.4622 - val_accuracy: 0.8402
Epoch 11/20
26/26 [==============================] - 6s 240ms/step - loss: 0.1661 - accuracy: 0.9548 - val_loss: 0.4506 - val_accuracy: 0.8574
Epoch 12/20
26/26 [==============================] - 8s 307ms/step - loss: 0.1313 - accuracy: 0.9657 - val_loss: 0.5129 - val_accuracy: 0.8536
Epoch 13/20
26/26 [==============================] - 6s 240ms/step - loss: 0.1097 - accuracy: 0.9717 - val_loss: 0.4424 - val_accuracy: 0.8658
Epoch 14/20
26/26 [==============================] - 8s 308ms/step - loss: 0.0948 - accuracy: 0.9758 - val_loss: 0.4525 - val_accuracy: 0.8639
Epoch 15/20
26/26 [==============================] - 7s 264ms/step - loss: 0.0798 - accuracy: 0.9801 - val_loss: 0.4716 - val_accuracy: 0.8642
Epoch 16/20
26/26 [==============================] - 9s 358ms/step - loss: 0.0667 - accuracy: 0.9845 - val_loss: 0.4871 - val_accuracy: 0.8620
Epoch 17/20
26/26 [==============================] - 7s 264ms/step - loss: 0.0604 - accuracy: 0.9859 - val_loss: 0.5010 - val_accuracy: 0.8670
Epoch 18/20
26/26 [==============================] - 7s 274ms/step - loss: 0.0505 - accuracy: 0.9896 - val_loss: 0.5322 - val_accuracy: 0.8655
Epoch 19/20
26/26 [==============================] - 7s 258ms/step - loss: 0.0455 - accuracy: 0.9900 - val_loss: 0.5673 - val_accuracy: 0.8639
Epoch 20/20
26/26 [==============================] - 7s 273ms/step - loss: 0.0423 - accuracy: 0.9916 - val_loss: 0.5089 - val_accuracy: 0.8670
```

```python
# Preprocess the test data
X_test_cleaned = X_test.apply(clean_text)

# Tokenize and pad the test data
test_sequences = tokenizer.texts_to_sequences(X_test_cleaned)
test_padded = pad_sequences(test_sequences, maxlen=max_length, padding='pre', truncating='pre')

# Predict sentiment labels for test data using model_LSTM
pred = model_LSTM.predict(test_padded)

# Convert predicted probabilities to predicted labels
predicted_labels = np.argmax(pred, axis=1)

# Generate classification report
print(classification_report(y_test, predicted_labels))
```

```
126/126 [==============================] - 3s 15ms/step
              precision    recall  f1-score   support

           0       0.73      0.75      0.74       730
           1       0.88      0.89      0.88      1395
           2       0.89      0.87      0.88      1889

    accuracy                           0.86      4014
   macro avg       0.83      0.84      0.83      4014
weighted avg       0.86      0.86      0.86      4014
```

```python
model_CNN = models.Sequential()
model_CNN.add(layers.Embedding(input_dim=vocab_size, output_dim=128, input_length=max_length))
model_CNN.add(layers.Conv1D(48, 7, activation='relu'))
model_CNN.add(layers.MaxPooling1D(5))
model_CNN.add(layers.Conv1D(48, 7, activation='relu'))
model_CNN.add(layers.GlobalMaxPooling1D())
model_CNN.add(layers.Dense(3, activation='softmax'))

# Print the summary of the model
model_CNN.summary()

# Compile the model
model_CNN.compile(optimizer='adam',
               loss='categorical_crossentropy',
               metrics=['accuracy'])
```

```
Model: "sequential_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_16 (Embedding)    (None, 90, 128)           1280000

 conv1d_2 (Conv1D)           (None, 84, 48)            43056

 max_pooling1d_1 (MaxPoolin  (None, 16, 48)            0
```

```
 g1D)

 conv1d_3 (Conv1D)           (None, 10, 48)            16176

 global_max_pooling1d_1 (Gl  (None, 48)                0
 obalMaxPooling1D)

 dense_26 (Dense)            (None, 3)                 147

=================================================================
Total params: 1339379 (5.11 MB)
Trainable params: 1339379 (5.11 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Fit the model
history = model_CNN.fit(train_padded,
                     y_train_encoded,
                     epochs=20,
                     batch_size=512,
                     validation_split=0.2)
```

```
Epoch 1/20
26/26 [==============================] - 13s 446ms/step - loss: 1.0307 - accuracy: 0.4639 - val_loss: 0.9760 - val_accuracy: 0.4840
Epoch 2/20
26/26 [==============================] - 10s 405ms/step - loss: 0.9228 - accuracy: 0.5865 - val_loss: 0.8154 - val_accuracy: 0.6487
Epoch 3/20
26/26 [==============================] - 10s 391ms/step - loss: 0.7428 - accuracy: 0.6911 - val_loss: 0.6983 - val_accuracy: 0.7119
Epoch 4/20
26/26 [==============================] - 9s 342ms/step - loss: 0.6041 - accuracy: 0.7631 - val_loss: 0.6631 - val_accuracy: 0.7359
Epoch 5/20
26/26 [==============================] - 10s 396ms/step - loss: 0.4950 - accuracy: 0.8217 - val_loss: 0.6545 - val_accuracy: 0.7533
Epoch 6/20
26/26 [==============================] - 11s 414ms/step - loss: 0.3891 - accuracy: 0.8686 - val_loss: 0.6226 - val_accuracy: 0.7720
Epoch 7/20
26/26 [==============================] - 9s 350ms/step - loss: 0.3197 - accuracy: 0.8937 - val_loss: 0.6334 - val_accuracy: 0.7783
Epoch 8/20
26/26 [==============================] - 11s 407ms/step - loss: 0.2691 - accuracy: 0.9130 - val_loss: 0.6832 - val_accuracy: 0.7664
Epoch 9/20
26/26 [==============================] - 11s 432ms/step - loss: 0.2391 - accuracy: 0.9214 - val_loss: 0.7314 - val_accuracy: 0.7618
Epoch 10/20
26/26 [==============================] - 10s 377ms/step - loss: 0.2126 - accuracy: 0.9306 - val_loss: 0.7541 - val_accuracy: 0.7664
Epoch 11/20
26/26 [==============================] - 10s 393ms/step - loss: 0.1936 - accuracy: 0.9366 - val_loss: 0.7767 - val_accuracy: 0.7593
Epoch 12/20
26/26 [==============================] - 10s 399ms/step - loss: 0.1759 - accuracy: 0.9421 - val_loss: 0.8277 - val_accuracy: 0.7655
Epoch 13/20
26/26 [==============================] - 10s 407ms/step - loss: 0.1642 - accuracy: 0.9457 - val_loss: 0.8690 - val_accuracy: 0.7621
Epoch 14/20
26/26 [==============================] - 9s 339ms/step - loss: 0.1524 - accuracy: 0.9511 - val_loss: 0.9048 - val_accuracy: 0.7583
Epoch 15/20
26/26 [==============================] - 10s 398ms/step - loss: 0.1398 - accuracy: 0.9532 - val_loss: 0.9435 - val_accuracy: 0.7546
Epoch 16/20
26/26 [==============================] - 11s 407ms/step - loss: 0.1288 - accuracy: 0.9581 - val_loss: 0.9793 - val_accuracy: 0.7627
Epoch 17/20
26/26 [==============================] - 9s 337ms/step - loss: 0.1201 - accuracy: 0.9600 - val_loss: 1.0223 - val_accuracy: 0.7533
Epoch 18/20
26/26 [==============================] - 10s 396ms/step - loss: 0.1136 - accuracy: 0.9625 - val_loss: 1.0568 - val_accuracy: 0.7499
Epoch 19/20
26/26 [==============================] - 11s 413ms/step - loss: 0.1099 - accuracy: 0.9640 - val_loss: 1.0898 - val_accuracy: 0.7518
Epoch 20/20
26/26 [==============================] - 11s 427ms/step - loss: 0.1034 - accuracy: 0.9650 - val_loss: 1.1333 - val_accuracy: 0.7468
```

```
# Preprocess the test data
X_test_cleaned = X_test.apply(clean_text)

# Tokenize and pad the test data
test_sequences = tokenizer.texts_to_sequences(X_test_cleaned)
test_padded = pad_sequences(test_sequences, maxlen=max_length, padding='pre', truncating='pre')

# Predict sentiment labels for test data using model_CNN
pred = model_CNN.predict(test_padded)

# Convert predicted probabilities to predicted labels
predicted_labels = np.argmax(pred, axis=1)

# Generate classification report
print(classification_report(y_test, predicted_labels))
```

```
126/126 [==============================] - 2s 15ms/step
              precision    recall  f1-score   support
```

```
              0       0.65      0.61      0.63       730
              1       0.72      0.77      0.75      1395
              2       0.78      0.76      0.77      1889

       accuracy                           0.74      4014
      macro avg       0.72      0.71      0.72      4014
   weighted avg       0.74      0.74      0.74      4014
```

The SimpleRNN model achieved an accuracy of 84%, with precision ranging from 78% to 87% across sentiment classes. It exhibited recall rates between 67% and 90%, with F1-scores varying from 0.72 to 0.86

The LSTM model achieved an accuracy of 86%, with precision ranging from 73% to 89% across sentiment classes. It exhibited recall rates between 75% and 89%, with F1-scores varying from 0.74 to 0.88

The CNN model achieved an accuracy of 74%, with precision ranging from 65% to 78% across sentiment classes. It exhibited recall rates between 61% and 77%, with F1-scores varying from 0.63 to 0.77

Therefore, based on the provided metrics, the LSTM model appears to be the best-performing model for this sentiment classification task.

```
# Download GloVe word embeddings
!wget https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip

# Extract the downloaded file
!unzip -q glove.6B.zip
```

```
--2024-04-05 00:04:34--  https://downloads.cs.stanford.edu/nlp/data/glove.6B.zip
Resolving downloads.cs.stanford.edu (downloads.cs.stanford.edu)... 171.64.64.22
Connecting to downloads.cs.stanford.edu (downloads.cs.stanford.edu)|171.64.64.22|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 862182613 (822M) [application/zip]
Saving to: 'glove.6B.zip'

glove.6B.zip        100%[===================>] 822.24M  5.06MB/s    in 2m 39s

2024-04-05 00:07:13 (5.17 MB/s) - 'glove.6B.zip' saved [862182613/862182613]
```

```
embeddings_index = {}
embedding_dim = 100  # Assuming the embedding dimension is 100
glove_path = 'glove.6B.100d.txt'  # Path to the GloVe embeddings file

# Load GloVe embeddings into embeddings_index dictionary
with open(glove_path, encoding="utf8") as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs

# Create an embedding matrix for words in the tokenizer's word index
embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, i in word_index.items():
    if i < vocab_size:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
```

```
from keras.models import Sequential
from keras.layers import Embedding, LSTM, Dense

# Define the model
model_lstm = Sequential()
model_lstm.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=max_length))
model_lstm.add(LSTM(32, dropout=0.2, recurrent_dropout=0.2))
model_lstm.add(Dense(3, activation='softmax'))
```

```
# Compile the model
model_lstm.compile(optimizer='adam',
               loss='categorical_crossentropy',
               metrics=['accuracy'])
```

```
# Fit the model
training_history = model_lstm.fit(train_padded,
                    y_train_encoded,
                    epochs=20,
                    batch_size=512,
                    validation_split=0.2)
```

```
Epoch 1/20
26/26 [==============================] - 30s 937ms/step - loss: 1.0337 - accuracy: 0.4762 - val_loss: 0.9593 - val_accuracy: 0.5297
Epoch 2/20
26/26 [==============================] - 25s 961ms/step - loss: 0.8905 - accuracy: 0.6015 - val_loss: 0.7887 - val_accuracy: 0.6450
Epoch 3/20
26/26 [==============================] - 21s 811ms/step - loss: 0.6675 - accuracy: 0.7134 - val_loss: 0.5893 - val_accuracy: 0.7708
Epoch 4/20
26/26 [==============================] - 19s 729ms/step - loss: 0.4303 - accuracy: 0.8471 - val_loss: 0.4603 - val_accuracy: 0.8265
Epoch 5/20
26/26 [==============================] - 24s 952ms/step - loss: 0.2734 - accuracy: 0.9161 - val_loss: 0.4118 - val_accuracy: 0.8577
Epoch 6/20
26/26 [==============================] - 23s 881ms/step - loss: 0.1833 - accuracy: 0.9456 - val_loss: 0.3758 - val_accuracy: 0.8739
Epoch 7/20
26/26 [==============================] - 19s 697ms/step - loss: 0.1327 - accuracy: 0.9643 - val_loss: 0.3509 - val_accuracy: 0.8888
Epoch 8/20
26/26 [==============================] - 22s 831ms/step - loss: 0.1049 - accuracy: 0.9725 - val_loss: 0.3455 - val_accuracy: 0.8932
Epoch 9/20
26/26 [==============================] - 24s 957ms/step - loss: 0.0847 - accuracy: 0.9780 - val_loss: 0.3493 - val_accuracy: 0.8954
Epoch 10/20
26/26 [==============================] - 20s 774ms/step - loss: 0.0782 - accuracy: 0.9817 - val_loss: 0.3712 - val_accuracy: 0.8969
Epoch 11/20
26/26 [==============================] - 19s 701ms/step - loss: 0.0633 - accuracy: 0.9843 - val_loss: 0.3815 - val_accuracy: 0.8954
Epoch 12/20
26/26 [==============================] - 25s 946ms/step - loss: 0.0525 - accuracy: 0.9878 - val_loss: 0.3802 - val_accuracy: 0.8975
Epoch 13/20
26/26 [==============================] - 23s 895ms/step - loss: 0.0456 - accuracy: 0.9899 - val_loss: 0.4200 - val_accuracy: 0.8916
Epoch 14/20
26/26 [==============================] - 18s 690ms/step - loss: 0.0406 - accuracy: 0.9908 - val_loss: 0.4070 - val_accuracy: 0.8935
Epoch 15/20
26/26 [==============================] - 21s 811ms/step - loss: 0.0362 - accuracy: 0.9921 - val_loss: 0.4248 - val_accuracy: 0.8891
Epoch 16/20
26/26 [==============================] - 24s 949ms/step - loss: 0.0333 - accuracy: 0.9926 - val_loss: 0.4330 - val_accuracy: 0.8926
Epoch 17/20
26/26 [==============================] - 20s 780ms/step - loss: 0.0311 - accuracy: 0.9924 - val_loss: 0.4394 - val_accuracy: 0.8898
Epoch 18/20
26/26 [==============================] - 18s 692ms/step - loss: 0.0308 - accuracy: 0.9932 - val_loss: 0.4631 - val_accuracy: 0.8938
Epoch 19/20
26/26 [==============================] - 24s 925ms/step - loss: 0.0266 - accuracy: 0.9938 - val_loss: 0.4573 - val_accuracy: 0.8919
Epoch 20/20
26/26 [==============================] - 24s 922ms/step - loss: 0.0228 - accuracy: 0.9951 - val_loss: 0.4760 - val_accuracy: 0.8919
```

```
# Preprocess the test data
X_test_cleaned = X_test.apply(clean_text)

# Tokenize and pad the test data
test_sequences = tokenizer.texts_to_sequences(X_test_cleaned)
test_padded = pad_sequences(test_sequences, maxlen=max_length, padding='pre', truncating='pre')

# Predict sentiment labels for test data using model_lstm
pred = model_lstm.predict(test_padded)

# Convert predicted probabilities to predicted labels
predicted_labels = np.argmax(pred, axis=1)

# Generate classification report
print(classification_report(y_test, predicted_labels))
```

```
126/126 [==============================] - 5s 33ms/step
              precision    recall  f1-score   support

           0       0.80      0.75      0.77       730
           1       0.89      0.93      0.91      1395
           2       0.91      0.90      0.90      1889

    accuracy                           0.88      4014
   macro avg       0.87      0.86      0.86      4014
weighted avg       0.88      0.88      0.88      4014
```

```
from gensim.models import Word2Vec
from gensim.test.utils import common_texts

# Train the Word2Vec model on common texts with different parameters
new_model = Word2Vec(sentences=common_texts, vector_size=300, window=10, min_count=2, sg=1, workers=4)
```

```
# Get the dimensionality of the word vectors in the Word2Vec model
embedding_dim = new_model.vector_size

# Initialize an embedding matrix with zeros
embedding_matrix = np.zeros((vocab_size, embedding_dim))

# Iterate over each word in the word index
for word, i in word_index.items():
    # Check if the index is within the maximum number of words
    if i < vocab_size:
        # Check if the word exists in the Word2Vec model's vocabulary
        if word in new_model.wv:
            # If the word exists, retrieve its word vector and assign it to the corresponding row in the embedding matrix
            embedding_matrix[i] = new_model.wv[word]
```

```
model_lstm = Sequential()
model_lstm.add(Embedding(input_dim=vocab_size, output_dim=embedding_dim, input_length=max_length))
model_lstm.add(LSTM(32, dropout=0.2, recurrent_dropout=0.2))
model_lstm.add(Dense(3, activation='softmax'))
```

```
# Compile the model
model_lstm.compile(optimizer='adam',
               loss='categorical_crossentropy',
               metrics=['accuracy'])
```

```
# Fit the model
history = model_lstm.fit(train_padded,y_train_encoded, epochs=10, batch_size=512, validation_split=0.2)
```

```
Epoch 1/10
26/26 [==============================] - 59s 2s/step - loss: 0.9961 - accuracy: 0.5391 - val_loss: 0.8714 - val_accuracy: 0.6179
Epoch 2/10
26/26 [==============================] - 49s 2s/step - loss: 0.7548 - accuracy: 0.6740 - val_loss: 0.6283 - val_accuracy: 0.7434
Epoch 3/10
26/26 [==============================] - 43s 2s/step - loss: 0.4602 - accuracy: 0.8264 - val_loss: 0.4557 - val_accuracy: 0.8250
Epoch 4/10
26/26 [==============================] - 47s 2s/step - loss: 0.2575 - accuracy: 0.9173 - val_loss: 0.3694 - val_accuracy: 0.8683
Epoch 5/10
26/26 [==============================] - 42s 2s/step - loss: 0.1482 - accuracy: 0.9593 - val_loss: 0.3548 - val_accuracy: 0.8851
Epoch 6/10
26/26 [==============================] - 50s 2s/step - loss: 0.1044 - accuracy: 0.9717 - val_loss: 0.3462 - val_accuracy: 0.8941
Epoch 7/10
26/26 [==============================] - 41s 2s/step - loss: 0.0774 - accuracy: 0.9813 - val_loss: 0.3504 - val_accuracy: 0.8954
Epoch 8/10
26/26 [==============================] - 50s 2s/step - loss: 0.0592 - accuracy: 0.9857 - val_loss: 0.3618 - val_accuracy: 0.9010
Epoch 9/10
26/26 [==============================] - 41s 2s/step - loss: 0.0499 - accuracy: 0.9880 - val_loss: 0.3877 - val_accuracy: 0.8975
Epoch 10/10
26/26 [==============================] - 51s 2s/step - loss: 0.0421 - accuracy: 0.9907 - val_loss: 0.3979 - val_accuracy: 0.8957
```

```
# Preprocess the test data
X_test_cleaned = X_test.apply(clean_text)
# Tokenize and pad the test data
test_sequences = tokenizer.texts_to_sequences(X_test_cleaned)
test_padded = pad_sequences(test_sequences, maxlen=max_length, padding='pre', truncating='pre')
# Predict sentiment labels for test data using model_lstm
pred = model_lstm.predict(test_padded)
# Convert predicted probabilities to predicted labels
predicted_labels = np.argmax(pred, axis=1)
# Generate classification report
print(classification_report(y_test, predicted_labels))
```

```
126/126 [==============================] - 9s 63ms/step
              precision    recall  f1-score   support

           0       0.82      0.75      0.79       730
           1       0.89      0.94      0.91      1395
           2       0.91      0.90      0.90      1889

    accuracy                           0.89      4014
   macro avg       0.87      0.86      0.87      4014
weighted avg       0.88      0.89      0.88      4014
```

The GloVe embeddings model achieved an accuracy of 88%, with precision ranging from 80% to 91% across sentiment classes. It exhibited recall rates between 75% and 93%, with F1-scores varying from 0.77 to 0.91

The Word2Vec model achieved an accuracy of 89%, with precision ranging from 82% to 91% across sentiment classes. It exhibited recall rates between 75% and 94%, with F1-scores varying from 0.79 to 0.91

From the above results Word2Vec model can be considered slightly superior to the GloVe embeddings model in this specific sentiment classification task, as it achieved slightly higher accuracy and performed slightly better across most performance metrics.