In [1]:

```
#import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report

import tensorflow as tf
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, LSTM, Dense, Dropout

#load the dataset
df = pd.read_csv('Tweets.csv')
```

In [5]:
```
#preview data
df.head (5)
```

Out[5]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline	airline
0	570306133677760513	neutral	1.0000	NaN	NaN	Virgin America	
1	570301130888122368	positive	0.3486	NaN	0.0000	Virgin America	
2	570301083672813571	neutral	0.6837	NaN	NaN	Virgin America	
3	570301031407624196	negative	1.0000	Bad Flight	0.7033	Virgin America	
4	570300817074462722	negative	1.0000	Can't Tell	1.0000	Virgin America	

In [2]:
```
#describe dataset
df.describe()
```

Out[2]:

	tweet_id	airline_sentiment_confidence	negativereason_confidence	retweet_count
count	1.464000e+04	14640.000000	10522.000000	14640.000000
mean	5.692184e+17	0.900169	0.638298	0.082650
std	7.791112e+14	0.162830	0.330440	0.745778
min	5.675883e+17	0.335000	0.000000	0.000000
25%	5.685592e+17	0.692300	0.360600	0.000000
50%	5.694779e+17	1.000000	0.670600	0.000000
75%	5.698905e+17	1.000000	1.000000	0.000000
max	5.703106e+17	1.000000	1.000000	44.000000

In [6]:
```
# remove unnecessary columns
df = df[['text', 'airline_sentiment']]

# clean the tweets
import re
def clean_text(text):
    text = re.sub(r'@[A-Za-z0-9]+', '', text) # remove mentions
    text = re.sub(r'https?://[A-Za-z0-9./]+', '', text) # remove urls
    text = re.sub(r'[^A-Za-z0-9\s]+', '', text) # remove special characters
    text = text.lower() # convert to lowercase
    return text

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

```
C:\Users\owner\AppData\Local\Temp\ipykernel_40308\3067754194.py:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  df['text'] = df['text'].apply(clean_text)
```

In [7]:
```
#Train-test split for model training
X_train, X_test, y_train, y_test = train_test_split(df['text'], df['airline_sentiment'], test_size=0.2, r
```

In [8]:
```
#Tokenize and Pad Sequences
tokenizer = Tokenizer(num_words=10000, oov_token="<OOV>")
tokenizer.fit_on_texts(X_train)

X_train_seq = tokenizer.texts_to_sequences(X_train)
X_train_pad = pad_sequences(X_train_seq, maxlen=100, padding='post', truncating='post')

X_test_seq = tokenizer.texts_to_sequences(X_test)
X_test_pad = pad_sequences(X_test_seq, maxlen=100, padding='post', truncating='post')
```

In [9]:
```
#build LSTM model
model = Sequential()

model.add(Embedding(10000, 128, input_length=100))
model.add(LSTM(64, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(3, activation='softmax'))

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

In [10]:
```
#train model
from tensorflow.keras.utils import to_categorical

y_train_cat = to_categorical(y_train.map({'negative': 0, 'neutral': 1, 'positive': 2}))
y_test_cat = to_categorical(y_test.map({'negative': 0, 'neutral': 1, 'positive': 2}))
```

In [11]:
```
history = model.fit(X_train_pad, y_train_cat, validation_data=(X_test_pad, y_test_cat), epochs=10, batch_
```

```
Epoch 1/10
92/92 [==============================] - 60s 578ms/step - loss: 0.9366 - accuracy: 0.6165 - val_loss: 0.
8952 - val_accuracy: 0.6452
Epoch 2/10
92/92 [==============================] - 51s 555ms/step - loss: 0.9219 - accuracy: 0.6224 - val_loss: 0.
9039 - val_accuracy: 0.6452
Epoch 3/10
92/92 [==============================] - 52s 566ms/step - loss: 0.9229 - accuracy: 0.6224 - val_loss: 0.
8943 - val_accuracy: 0.6452
Epoch 4/10
92/92 [==============================] - 52s 564ms/step - loss: 0.9221 - accuracy: 0.6224 - val_loss: 0.
8956 - val_accuracy: 0.6452
Epoch 5/10
92/92 [==============================] - 52s 563ms/step - loss: 0.9223 - accuracy: 0.6224 - val_loss: 0.
8977 - val_accuracy: 0.6452
Epoch 6/10
92/92 [==============================] - 52s 561ms/step - loss: 0.9219 - accuracy: 0.6224 - val_loss: 0.
8959 - val_accuracy: 0.6452
Epoch 7/10
92/92 [==============================] - 52s 565ms/step - loss: 0.9218 - accuracy: 0.6224 - val_loss: 0.
8984 - val_accuracy: 0.6452
Epoch 8/10
92/92 [==============================] - 52s 560ms/step - loss: 0.9222 - accuracy: 0.6224 - val_loss: 0.
8958 - val_accuracy: 0.6452
Epoch 9/10
92/92 [==============================] - 52s 564ms/step - loss: 0.9222 - accuracy: 0.6224 - val_loss: 0.
8974 - val_accuracy: 0.6452
Epoch 10/10
92/92 [==============================] - 52s 561ms/step - loss: 0.9220 - accuracy: 0.6224 - val_loss: 0.
8943 - val_accuracy: 0.6452
```

In [12]:

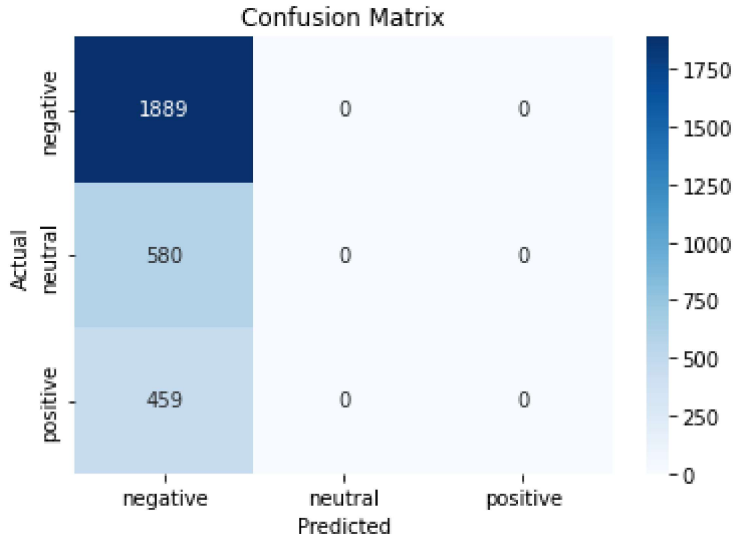
```python
# classification report
y_pred = np.argmax(model.predict(X_test_pad), axis=-1)

print(classification_report(y_test.map({'negative': 0, 'neutral': 1, 'positive': 2}), y_pred))
```

```
92/92 [==============================] - 4s 36ms/step
              precision    recall  f1-score   support

           0       0.65      1.00      0.78      1889
           1       0.00      0.00      0.00       580
           2       0.00      0.00      0.00       459

    accuracy                           0.65      2928
   macro avg       0.22      0.33      0.26      2928
weighted avg       0.42      0.65      0.51      2928


C:\Users\owner\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\owner\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\owner\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarni
ng: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [13]:

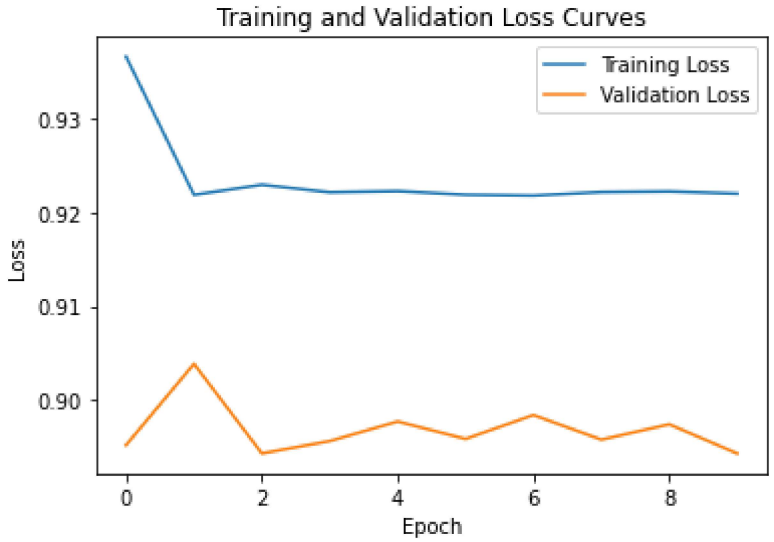
```
# get the predicted labels
y_pred = np.argmax(model.predict(X_test_pad), axis=-1)

# create the confusion matrix
cm = confusion_matrix(y_test.map({'negative': 0, 'neutral': 1, 'positive': 2}), y_pred)

# plot the confusion matrix
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', xticklabels=['negative', 'neutral', 'positive'], ytick

plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
92/92 [==============================] - 4s 40ms/step
```



In [14]:

```
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss Curves')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



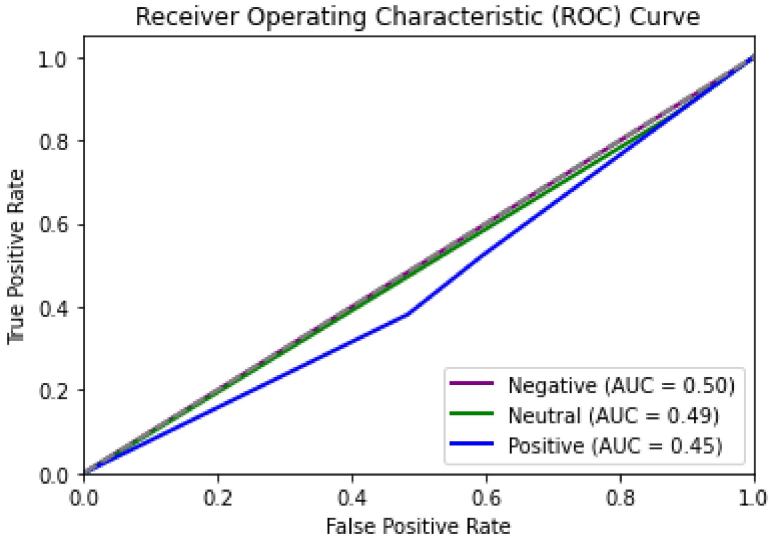
In [19]:

```
from sklearn.preprocessing import LabelEncoder

# encode the target variable
le = LabelEncoder()
y_test_encoded = le.fit_transform(y_test)
```

In [21]:

```
from sklearn.metrics import roc_curve, auc

# get the predicted probabilities for each class
y_pred_prob = model.predict(X_test_pad)

# encode the target variable
le = LabelEncoder()
y_test_encoded = le.fit_transform(y_test)

# calculate the fpr and tpr for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(3):
    fpr[i], tpr[i], _ = roc_curve((y_test_encoded == i).astype(int), y_pred_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# plot the ROC curves
plt.figure()
lw = 2
plt.plot(fpr[0], tpr[0], color='purple', lw=lw, label='Negative (AUC = %0.2f)' % roc_auc[0])
plt.plot(fpr[1], tpr[1], color='green', lw=lw, label='Neutral (AUC = %0.2f)' % roc_auc[1])
plt.plot(fpr[2], tpr[2], color='blue', lw=lw, label='Positive (AUC = %0.2f)' % roc_auc[2])
plt.plot([0, 1], [0, 1], color='gray', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()
```

```
92/92 [==============================] - 3s 37ms/step
```


Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Negative (AUC = 0.50)
Neutral (AUC = 0.49)
Positive (AUC = 0.45)

In [ ]: