US Airline Sentiment Analysis Using Deep Learning

By: Lathif Ramadhan

Project Description

This project aims to analyze the sentiment of tweets towards US airlines using Natural Language Processing (NLP) techniques and a Deep Learning model, specifically Long Short-Term Memory (LSTM). The dataset used is `Tweets.csv` from Kaggle, which contains tweets about various US airlines. The sentiment analysis results are expected to provide insights into public perception of airlines based on Twitter data.

Dataset Link: https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment/data

Step 1: Import Libraries and Set Up

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
import re
import nltk
import pickle
from nltk.corpus import stopwords
from tensorflow.keras.callbacks import EarlyStopping # Import
EarlyStopping

# Additional Setup
plt.style.use('seaborn-v0_8-whitegrid')

# Ensure NLTK punkt and stopwords are downloaded
try:
    nltk.data.find('tokenizers/punkt')
except LookupError:
    nltk.download('punkt')
try:
    nltk.data.find('corpora/stopwords')
except LookupError:
```

```
    nltk.download('stopwords')
try:
    nltk.data.find('tokenizers/punkt_tab') # to check for punkt_tab
except LookupError:
    nltk.download('punkt_tab') # to download punkt_tab


print("Setup complete and libraries imported successfully.")
```

```
[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 punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.

Setup complete and libraries imported successfully.
```

Step 2: Data Loading and Exploratory Data Analysis (EDA)

Load the dataset

```
df =
pd.read_csv('https://raw.githubusercontent.com/LatiefDataVisionary/
airline-sentiment-nlp-capstone/refs/heads/main/data/raw/Tweets.csv')
```

Display the first 5 rows of the dataframe

```
display(df.head())
```

```
{"repr_error":"Out of range float values are not JSON compliant:
nan","type":"dataframe"}
```

Basic Information

Display the shape of the dataframe

```
df.shape
```

```
(14640, 15)
```

Display basic information about the dataframe

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14640 entries, 0 to 14639
```

```
Data columns (total 15 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   tweet_id                       14640 non-null  int64
 1   airline_sentiment              14640 non-null  object
 2   airline_sentiment_confidence   14640 non-null  float64
 3   negativereason                 9178 non-null   object
 4   negativereason_confidence      10522 non-null  float64
 5   airline                        14640 non-null  object
 6   airline_sentiment_gold         40 non-null     object
 7   name                           14640 non-null  object
 8   negativereason_gold            32 non-null     object
 9   retweet_count                  14640 non-null  int64
 10  text                           14640 non-null  object
 11  tweet_coord                    1019 non-null   object
 12  tweet_created                  14640 non-null  object
 13  tweet_location                 9907 non-null   object
 14  user_timezone                  9820 non-null   object
dtypes: float64(2), int64(2), object(11)
memory usage: 1.7+ MB
```

Display descriptive statistics

```
display(df.describe(include='all'))
```


```
{"summary":"{\n  \"name\": \"display(df\",\n  \"rows\": 11,\n
\"fields\": [\n    {\n      \"column\": \"tweet_id\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
2.6329747500473715e+17,\n        \"min\": 14640.0,\n        \"max\":
5.703106004605256e+17,\n        \"num_unique_values\": 8,\n
\"samples\": [\n          5.692183517674992e+17,\n
5.694778579231109e+17,\n          14640.0\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"airline_sentiment\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n          3,\n
\"9178\",\n          \"14640\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"airline_sentiment_confidence\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
5175.764549399977,\n        \"min\": 0.1628299590986659,\n
\"max\": 14640.0,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          14640.0,\n          0.9001688524590163,\n
1.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"negativereason\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          10,\n          \"2910\",\n          \"9178\"\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
```

```
}\n    },\n    {\n      \"column\": \"negativereason_confidence\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
3719.8867640100684,\n        \"min\": 0.0,\n        \"max\": 10522.0,\
n        \"num_unique_values\": 7,\n        \"samples\": [\n
10522.0,\n          0.6382982797947159,\n          0.6706\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"airline\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n          6,\n
\"3822\",\n          \"14640\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"airline_sentiment_gold\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 4,\n
\"samples\": [\n          3,\n          \"32\",\n          \"40\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"name\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
4,\n        \"samples\": [\n          7701,\n          \"63\",\n
\"14640\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"negativereason_gold\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          13,\n          \"12\",\n          \"32\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"retweet_count\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
5173.780233286849,\n        \"min\": 0.0,\n        \"max\": 14640.0,\n
\"num_unique_values\": 5,\n        \"samples\": [\n
0.08265027322404371,\n          44.0,\n          0.7457781608465677\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"text\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
4,\n        \"samples\": [\n          14427,\n          \"6\",\n
\"14640\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"tweet_coord\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 4,\n        \"samples\":
[\n          832,\n          \"164\",\n          \"1019\"\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"tweet_created\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n          14247,\n
\"5\",\n          \"14640\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"tweet_location\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 4,\n
\"samples\": [\n          3081,\n          \"157\",\n
\"9907\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
```

\"user_timezone\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 4,\n   \"samples\":
[\n   85,\n   \"3744\",\n   \"9820\"\n   ],\
n   \"semantic_type\": \"\",\n   \"description\": \"\"\n
}\n   }\n]\n}","type":"dataframe"}

Check for missing values

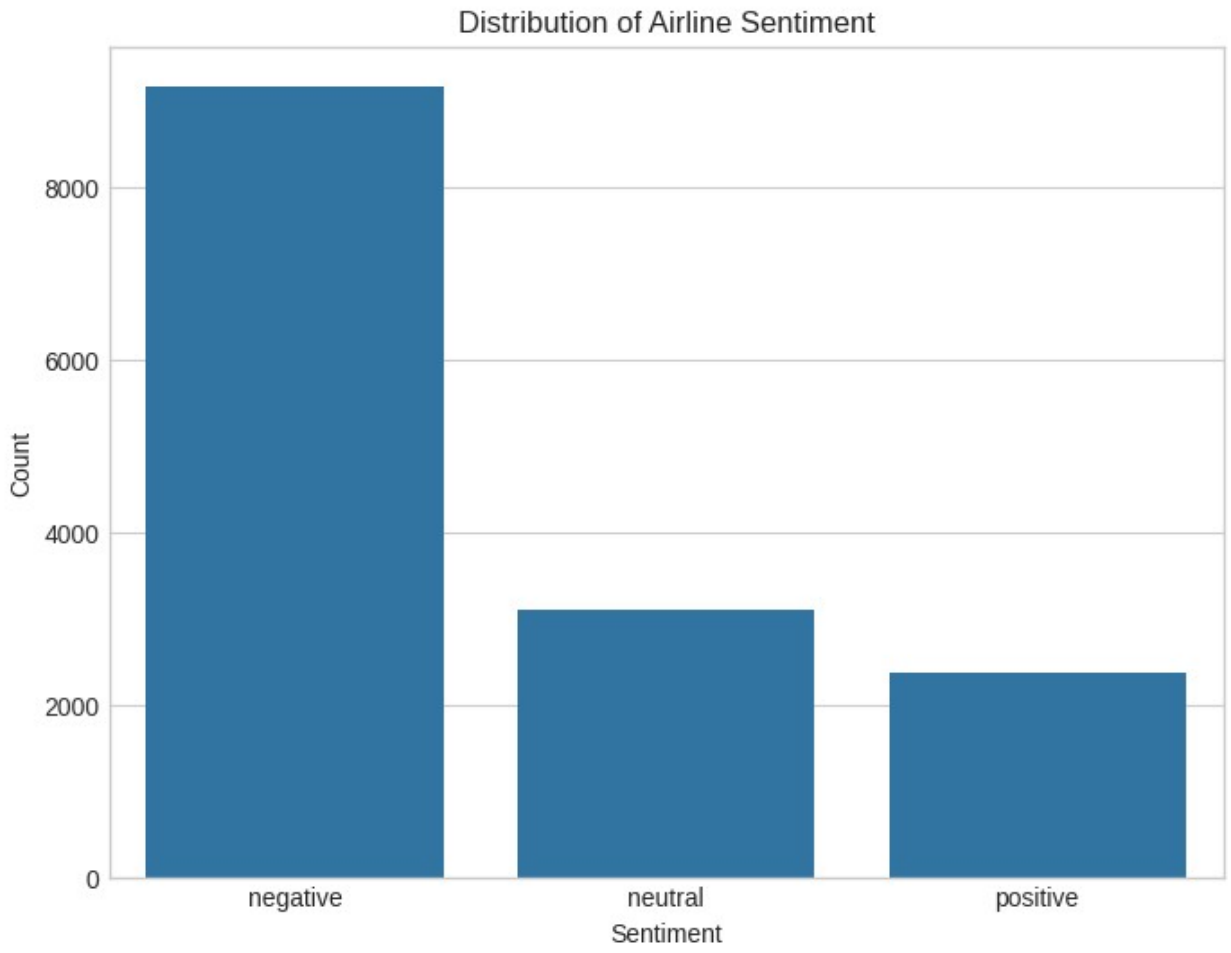
```
df.isnull().sum()
```

```
tweet_id                          0
airline_sentiment                 0
airline_sentiment_confidence      0
negativereason                 5462
negativereason_confidence      4118
airline                           0
airline_sentiment_gold        14600
name                              0
negativereason_gold           14608
retweet_count                     0
text                              0
tweet_coord                   13621
tweet_created                     0
tweet_location                 4733
user_timezone                  4820
dtype: int64
```

Data Visualization

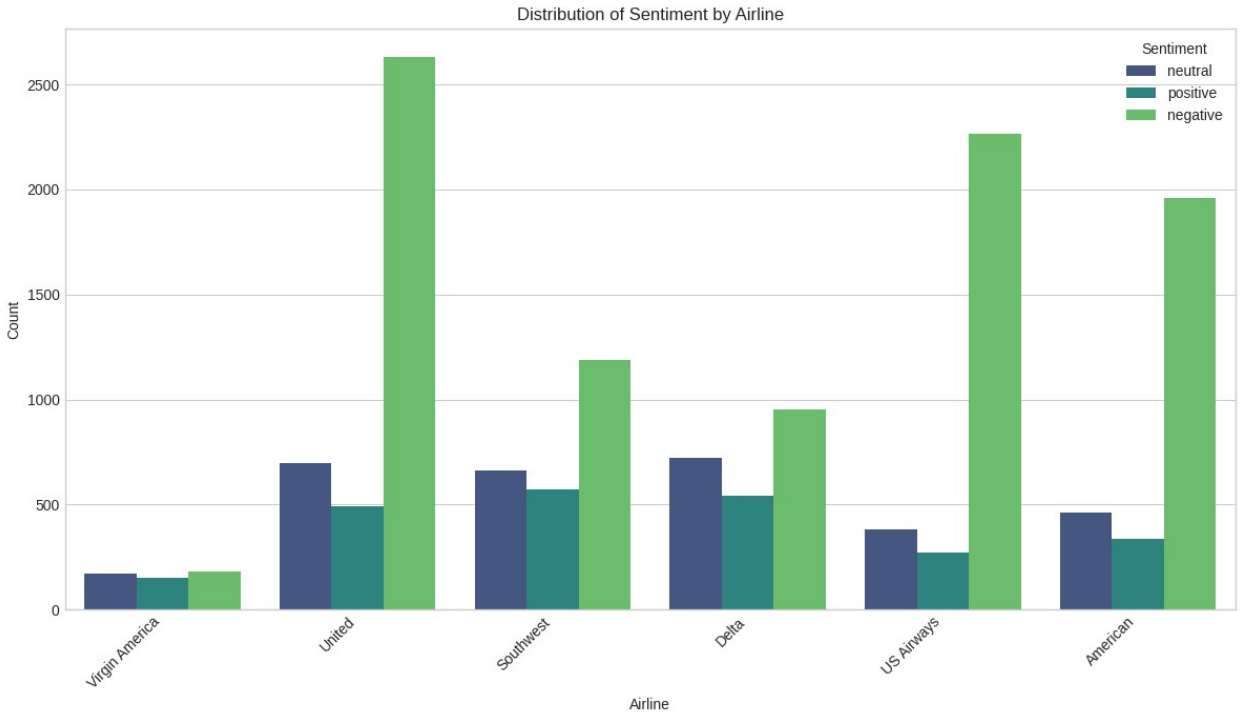
Visualize the distribution of airline_sentiment

```
plt.figure(figsize=(8, 6))
sns.countplot(data=df, x='airline_sentiment',
order=df['airline_sentiment'].value_counts().index)
plt.title('Distribution of Airline Sentiment')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.show()
```



Visualize the distribution of sentiment for each airline

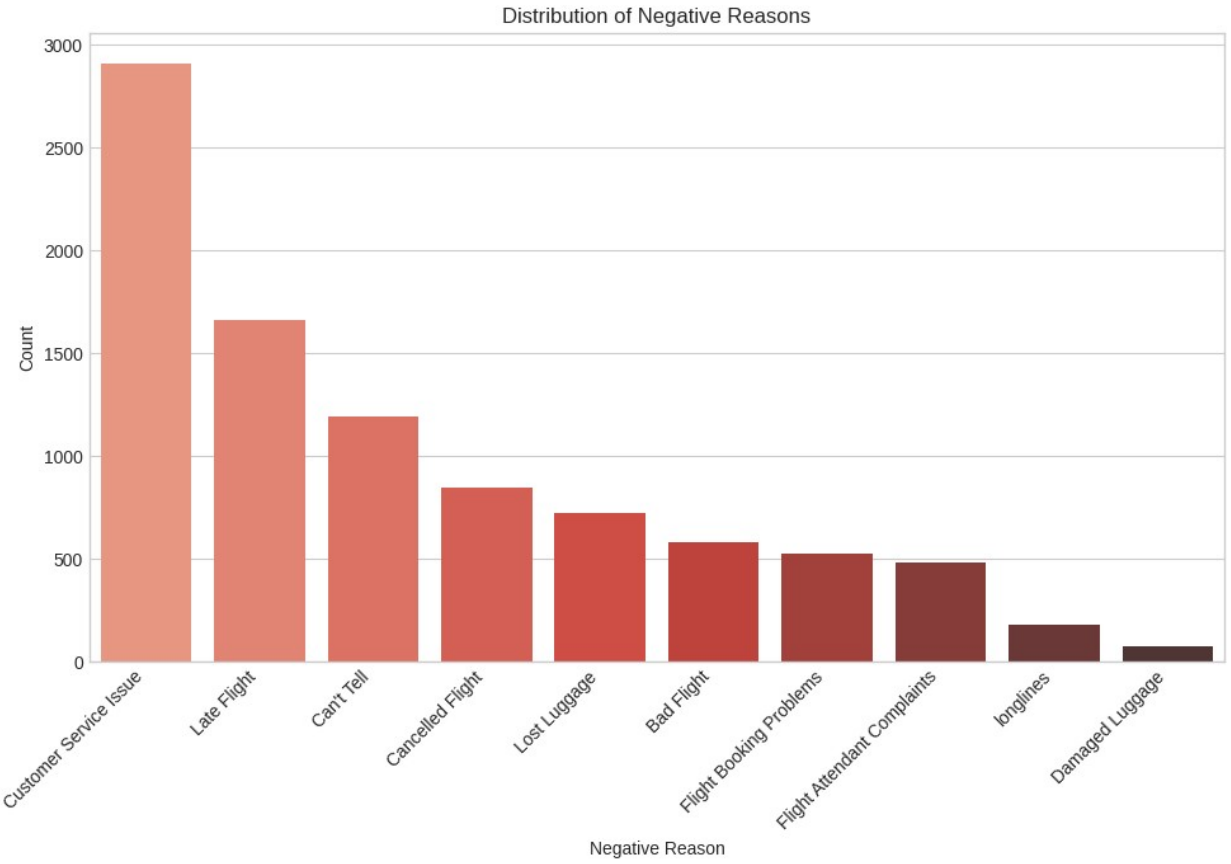
```
plt.figure(figsize=(12, 7))
sns.countplot(data=df, x='airline', hue='airline_sentiment',
palette='viridis')
plt.title('Distribution of Sentiment by Airline')
plt.xlabel('Airline')
plt.ylabel('Count')
plt.xticks(rotation=45, ha='right')
plt.legend(title='Sentiment')
plt.tight_layout()
plt.show()
```



Let's visualize the distribution of negative reasons for the negative sentiment tweets.

```
negative_reasons = df[df['airline_sentiment'] == 'negative']
['negativereason'].value_counts()

plt.figure(figsize=(10, 7))
sns.barplot(x=negative_reasons.index, y=negative_reasons.values,
palette='Reds_d')
plt.title('Distribution of Negative Reasons')
plt.xlabel('Negative Reason')
plt.ylabel('Count')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

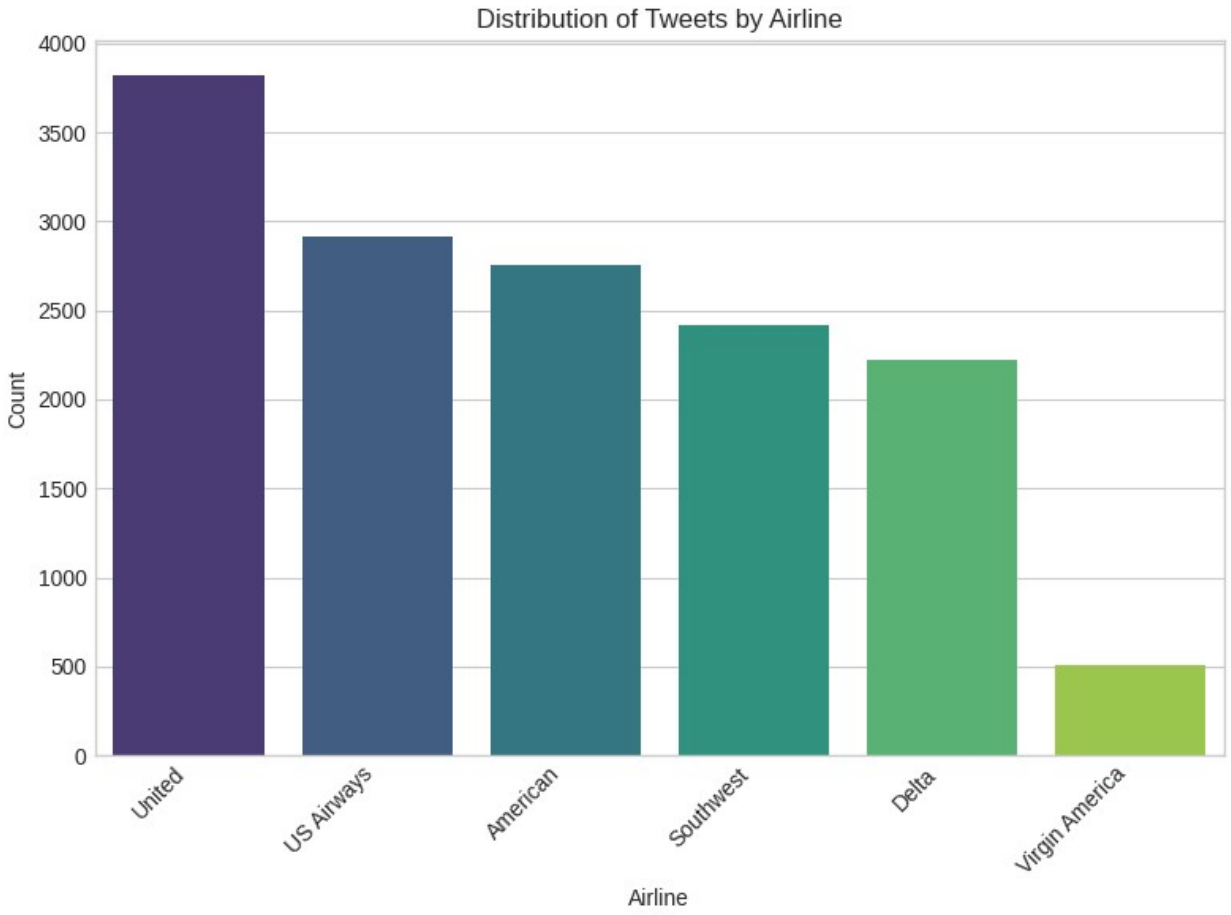
```
/tmp/ipython-input-777774778.py:4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.barplot(x=negative_reasons.index, y=negative_reasons.values,
palette='Reds_d')
```



Let's also visualize the distribution of tweets across different airlines.

```
plt.figure(figsize=(8, 6))
sns.countplot(data=df, x='airline',
order=df['airline'].value_counts().index, palette='viridis')
plt.title('Distribution of Tweets by Airline')
plt.xlabel('Airline')
plt.ylabel('Count')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-536558326.py:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be
removed in v0.14.0. Assign the `x` variable to `hue` and set
`legend=False` for the same effect.

  sns.countplot(data=df, x='airline',
order=df['airline'].value_counts().index, palette='viridis')
```



Step 3: Text Preprocessing & Cleaning

```
# Text Preprocessing Function
def preprocess_text(text):
    # Remove URLs
    text = re.sub(r'http\S+', '', text)
    # Remove mentions (@users) and hashtags (#)
    text = re.sub(r'@\w+|#', '', text)
    # Convert to lowercase
    text = text.lower()
    # Remove punctuation
    text = re.sub(r'[^\w\s]', '', text)
    # Remove numbers
    text = re.sub(r'\d+', '', text)
    # Tokenization
    tokens = nltk.word_tokenize(text)
    # Remove Stopwords
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]
    # Join tokens back into a string
    cleaned_text = ' '.join(tokens)
    return cleaned_text
```

```
# Apply the preprocessing function to the 'text' column
df['clean_text'] = df['text'].apply(preprocess_text)

# Display the first few rows of 'text' and 'clean_text' for comparison
print("Original vs Cleaned Text:")
display(df[['text', 'clean_text']].head(10))
```

```
Original vs Cleaned Text:
```


```
{"summary":"{\n  \"name\": \"display(df[['text', 'clean_text']]\",\n  \"rows\": 10,\n  \"fields\": [\n    {\n      \"column\": \"text\",\n      \"properties\": {\n        \"dtype\": \"string\",\n        \"num_unique_values\": 10,\n        \"samples\": [\n          \"@virginamerica Well, I didn't\\u2026but NOW I DO! :-D\",\n          \"@VirginAmerica plus you've added commercials to the experience... tacky.\",\n          \"@VirginAmerica seriously would pay $30 a flight for seats that didn't have this playing.\\nit's really the only bad thing about flying VA\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"clean_text\",\n      \"properties\": {\n        \"dtype\": \"string\",\n        \"num_unique_values\": 10,\n        \"samples\": [\n          \"well didntbut\",\n          \"plus youve added commercials experience tacky\",\n          \"seriously would pay flight seats didnt playing really bad thing flying va\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


Save Processed Data

Save the DataFrame with the cleaned text to a CSV file.

```
# Define the path to save the processed data
processed_data_path = 'Tweets_processed.csv'

# Save the DataFrame to a CSV file
# index=False prevents pandas from writing the DataFrame index as a column in the CSV
df.to_csv(processed_data_path, index=False)

print(f"Processed data saved successfully to '{processed_data_path}'")
```

```
Processed data saved successfully to 'Tweets_processed.csv'
```

Step 4: Data Preparation for Model

Define Features (X) and Target (y)

```
X = df['clean_text']
y = df['airline_sentiment']
```

Encoding Label Target (One-Hot Encoding)

```
y_encoded = pd.get_dummies(y)
```

Display the first few rows of the encoded target

```
y_encoded.head()
```


```
{"summary":"{\n  \"name\": \"y_encoded\",\n  \"rows\": 14640,\n  \"fields\": [\n    {\n      \"column\": \"negative\",\n      \"properties\": {\n        \"dtype\": \"boolean\",\n        \"num_unique_values\": 2,\n        \"samples\": [\n          true,\n          false\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"neutral\",\n      \"properties\": {\n        \"dtype\": \"boolean\",\n        \"num_unique_values\": 2,\n        \"samples\": [\n          false,\n          true\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"positive\",\n      \"properties\": {\n        \"dtype\": \"boolean\",\n        \"num_unique_values\": 2,\n        \"samples\": [\n          true,\n          false\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe","variable_name":"y_encoded"}
```


Split Data into Training and Testing Sets

```
# Split Data into Training and Testing Sets
# Use y_encoded (one-hot encoded labels) instead of original y
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)
```

Shape of training data (X_train, y_train)

```
print(X_train.shape, y_train.shape)
```

```
(11712,) (11712, 3)
```

Shape of testing data (X_test, y_test)

```
print(X_test.shape, y_test.shape)
```

```
(2928,) (2928, 3)
```

Tokenization and Sequencing

```
num_words = 10000
oov_token = "<OOV>"
tokenizer = Tokenizer(num_words=num_words, oov_token=oov_token)

# Fit tokenizer only on training data
tokenizer.fit_on_texts(X_train)

# Convert text to sequences
train_sequences = tokenizer.texts_to_sequences(X_train)
test_sequences = tokenizer.texts_to_sequences(X_test)

# Display a sample of sequences
print("\nSample of training sequences:")
print(train_sequences[0][:10])
print(test_sequences[0][:10])
```

```
Sample of training sequences:
[830, 11, 1606, 66, 241]
[131, 286, 7341, 134, 36, 828]
```

Padding Sequences

```
# Determine maxlen from the longest sequence in the training set
maxlen = max([len(x) for x in train_sequences])
print(f'\nMaximum sequence length in training data: {maxlen}')

X_train_pad = pad_sequences(train_sequences, maxlen=maxlen,
padding='post', truncating='post')
X_test_pad = pad_sequences(test_sequences, maxlen=maxlen,
padding='post', truncating='post')

print('\nShape of padded training sequences (X_train_pad):')
print(X_train_pad.shape)
print('\nShape of padded testing sequences (X_test_pad):')
print(X_test_pad.shape)

# Display a sample of padded sequences
print("\nSample of padded training sequences:")
print(X_train_pad[0][:10])
```

```
Maximum sequence length in training data: 21

Shape of padded training sequences (X_train_pad):
(11712, 21)

Shape of padded testing sequences (X_test_pad):
(2928, 21)
```

```
Sample of padded training sequences:
[ 830   11 1606   66  241    0    0    0    0    0]
```

Step 5: Building the Deep Learning Model (LSTM)

```
# Model Parameters
vocab_size = num_words  # Use the number of words from the tokenizer
embedding_dim = 100
max_length = maxlen # Use the maxlen from padding

# Model Architecture
model = keras.Sequential([
    Embedding(input_dim=vocab_size, output_dim=embedding_dim,
input_length=max_length),
    LSTM(64, dropout=0.2, recurrent_dropout=0.2),
    Dropout(0.5),
    Dense(3, activation='softmax') # 3 units for positive, neutral,
negative sentiments
])

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

# Model Summary
print("Model Architecture Summary:")
model.summary()
```

```
Model Architecture Summary:

/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/
embedding.py:97: UserWarning: Argument `input_length` is deprecated.
Just remove it.
  warnings.warn(

Model: "sequential"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━
┓
┃ Layer (type)                    ┃ Output Shape           ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━
┩
│ embedding (Embedding)           │ ?                      │   0
(unbuilt) │
├─────────────────────────────────┼────────────────────────┼───────────────
┤
│ lstm (LSTM)                     │ ?                      │   0
(unbuilt) │
├─────────────────────────────────┼────────────────────────┼───────────────
```

```
──────┤
| dropout (Dropout)                 | ?                         |
0 |
├─────────────────────────────────┼──────────────────────────┼──────────
──────┤
| dense (Dense)                   | ?                         |   0
(unbuilt) |
└─────────────────────────────────┴──────────────────────────┴──────────
──────┘

 Total params: 0 (0.00 B)

 Trainable params: 0 (0.00 B)

 Non-trainable params: 0 (0.00 B)
```

Step 6: Model Training and Evaluation

Model Training

```
epochs = 10  # You can adjust the number of epochs
batch_size = 32 # You can adjust the batch size

# Implement Early Stopping
# Monitor validation loss and stop training if it doesn't improve for
a few epochs
early_stopping = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)

print('Starting model training with Early Stopping...')
history = model.fit(X_train_pad, y_train,
                    epochs=epochs,
                    batch_size=batch_size,
                    validation_split=0.2, # Use a validation split
from the training data
                    callbacks=[early_stopping], # Add the
EarlyStopping callback
                    verbose=1)
print('Model training finished.')
```

```
Starting model training with Early Stopping...
Epoch 1/10
293/293 ━━━━━━━━━━━━━━━━━━━━ 20s 50ms/step - accuracy: 0.6269 - loss:
0.8973 - val_accuracy: 0.7384 - val_loss: 0.6479
Epoch 2/10
293/293 ━━━━━━━━━━━━━━━━━━━━ 13s 46ms/step - accuracy: 0.7652 - loss:
0.5684 - val_accuracy: 0.7776 - val_loss: 0.5714
Epoch 3/10
293/293 ━━━━━━━━━━━━━━━━━━━━ 20s 46ms/step - accuracy: 0.8569 - loss:
0.4027 - val_accuracy: 0.7687 - val_loss: 0.5965
```

```
Epoch 4/10
293/293 ━━━━━━━━━━━━━━━━━━━━ 21s 46ms/step - accuracy: 0.8992 - loss:
0.3060 - val_accuracy: 0.7623 - val_loss: 0.7034
Epoch 5/10
293/293 ━━━━━━━━━━━━━━━━━━━━ 21s 46ms/step - accuracy: 0.9255 - loss:
0.2422 - val_accuracy: 0.7529 - val_loss: 0.7187
Model training finished.
```

Model Evaluation

```
print("\nEvaluating model on the test set...")
loss, accuracy = model.evaluate(X_test_pad, y_test, verbose=0)

print(f"\nTest Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")
```

```
Evaluating model on the test set...

Test Loss: 0.5434
Test Accuracy: 0.7920
```

Generate predictions

```
y_pred_probs = model.predict(X_test_pad)
y_pred = np.argmax(y_pred_probs, axis=1)
y_test_labels = np.argmax(y_test.values, axis=1) # Convert one-hot
encoded test labels back
```

```
92/92 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step
```

Display Classification Report

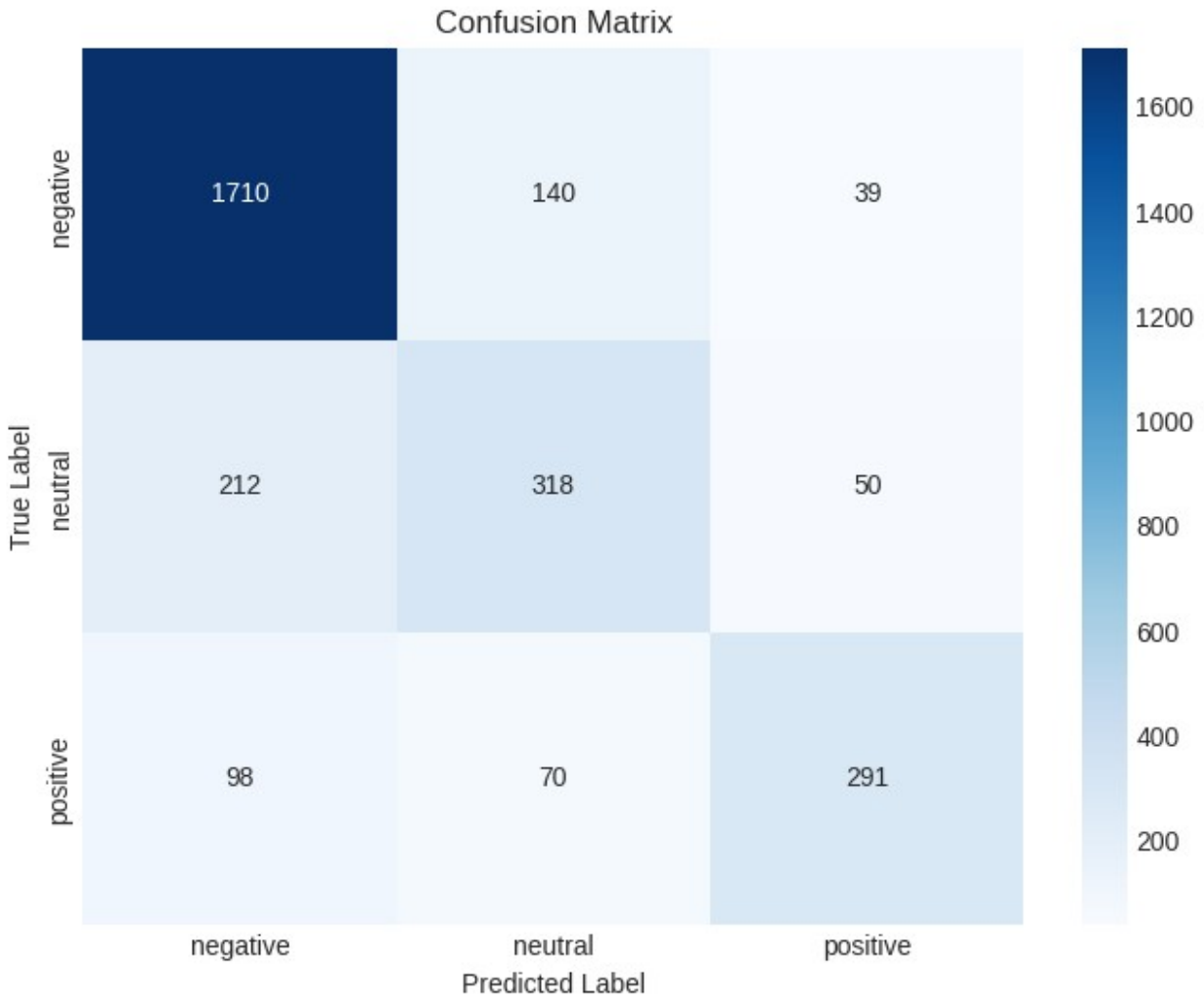
```
print("\nClassification Report:")
print(classification_report(y_test_labels, y_pred,
target_names=y_encoded.columns))
```

```
Classification Report:
              precision    recall  f1-score   support

    negative       0.85      0.91      0.87      1889
     neutral       0.60      0.55      0.57       580
    positive       0.77      0.63      0.69       459

    accuracy                           0.79      2928
   macro avg       0.74      0.70      0.71      2928
weighted avg       0.79      0.79      0.79      2928
```

Display Confusion Matrix

```
print("\nConfusion Matrix:")
cm = confusion_matrix(y_test_labels, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=y_encoded.columns, yticklabels=y_encoded.columns)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix:
```



Step 7: Save model and tokenizer

Save the trained LSTM model and the fitted tokenizer to files for later use.

```
# Save the model
model.save('lstm_sentiment_model.h5')
print("Trained LSTM model saved as 'lstm_sentiment_model.h5'")

# Save the tokenizer
with open('tokenizer.pickle', 'wb') as handle:
    pickle.dump(tokenizer, handle, protocol=pickle.HIGHEST_PROTOCOL)
print("Fitted tokenizer saved as 'tokenizer.pickle'")
```

```
WARNING:absl:You are saving your model as an HDF5 file via
`model.save()` or `keras.saving.save_model(model)`. This file format
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.

Trained LSTM model saved as 'lstm_sentiment_model.h5'
Fitted tokenizer saved as 'tokenizer.pickle'
```

Step 8: Making Predictions

Demonstrating how to make predictions on new, unseen text data using the loaded model and tokenizer.

Load the saved model and tokenizer, define new text samples, preprocess them, convert to sequences, pad the sequences, make predictions, map predictions back to labels, and print the results to demonstrate predictions on new data.

```
# Load the saved model
loaded_model = keras.models.load_model('lstm_sentiment_model.h5')
print("Loaded LSTM model from 'lstm_sentiment_model.h5'")

# Load the saved tokenizer
with open('tokenizer.pickle', 'rb') as handle:
    loaded_tokenizer = pickle.load(handle)
print("Loaded tokenizer from 'tokenizer.pickle'")

# Define new text samples
new_tweets = [
    "This airline service was excellent! Very friendly staff.",
    "My flight was delayed for 3 hours. Terrible experience.",
    "The flight was okay, nothing special.",
    "Had a great flight with comfortable seats and good
entertainment.",
    "Lost my luggage. Very disappointed with the handling."
]

# Preprocess the new text samples
preprocessed_new_tweets = [preprocess_text(tweet) for tweet in
new_tweets]
print("\nPreprocessed new tweets:")
```

```
for original, preprocessed in zip(new_tweets,
preprocessed_new_tweets):
    print(f"Original: '{original}'\nPreprocessed: '{preprocessed}'\n")

# Convert preprocessed text to sequences using the loaded tokenizer
new_sequences =
loaded_tokenizer.texts_to_sequences(preprocessed_new_tweets)

# Pad the sequences to the maximum length determined during training
# Use the maxlen variable from the previous steps
X_new_pad = pad_sequences(new_sequences, maxlen=maxlen,
padding='post', truncating='post')

print(f"\nShape of padded new sequences (X_new_pad):
{X_new_pad.shape}")

# Make predictions on the padded sequences
new_predictions_probs = loaded_model.predict(X_new_pad)

# Determine the predicted sentiment label for each sample
predicted_labels_index = np.argmax(new_predictions_probs, axis=1)

# Map indices back to sentiment labels
sentiment_map = {0: 'negative', 1: 'neutral', 2: 'positive'} # Based
on y_encoded columns order
predicted_sentiments = [sentiment_map[index] for index in
predicted_labels_index]

# Print the original text samples and their corresponding predicted
sentiment labels
print("\nPredictions on new tweets:")
for original_tweet, predicted_sentiment in zip(new_tweets,
predicted_sentiments):
    print(f"Tweet: '{original_tweet}'")
    print(f"Predicted Sentiment: {predicted_sentiment}\n")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have
yet to be built. `model.compile_metrics` will be empty until you train
or evaluate the model.

Loaded LSTM model from 'lstm_sentiment_model.h5'
Loaded tokenizer from 'tokenizer.pickle'

Preprocessed new tweets:
Original: 'This airline service was excellent! Very friendly staff.'
Preprocessed: 'airline service excellent friendly staff'

Original: 'My flight was delayed for 3 hours. Terrible experience.'
Preprocessed: 'flight delayed hours terrible experience'

Original: 'The flight was okay, nothing special.'
```

```
Preprocessed: 'flight okay nothing special'

Original: 'Had a great flight with comfortable seats and good
entertainment.'
Preprocessed: 'great flight comfortable seats good entertainment'

Original: 'Lost my luggage. Very disappointed with the handling.'
Preprocessed: 'lost luggage disappointed handling'


Shape of padded new sequences (X_new_pad): (5, 21)
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 544ms/step

Predictions on new tweets:
Tweet: 'This airline service was excellent! Very friendly staff.'
Predicted Sentiment: positive

Tweet: 'My flight was delayed for 3 hours. Terrible experience.'
Predicted Sentiment: negative

Tweet: 'The flight was okay, nothing special.'
Predicted Sentiment: negative

Tweet: 'Had a great flight with comfortable seats and good
entertainment.'
Predicted Sentiment: positive

Tweet: 'Lost my luggage. Very disappointed with the handling.'
Predicted Sentiment: negative
```

Step 9: Summarize findings

Interpreting the model's performance based on the evaluation metrics (accuracy, precision, recall, F1-score) and the confusion matrix. Discuss insights gained from the sentiment analysis.

```python
# Interpretation of Classification Report and Confusion Matrix

print("Interpretation of Model Performance:")

# Analyze Classification Report
print("\nAnalysis of Classification Report:")
print(
    "The classification report provides precision, recall, and F1-
score for each sentiment class."
)
print("- **Precision** is the ability of the classifier not to label
as positive a sample that is negative.")
print("- **Recall** is the ability of the classifier to find all the
positive samples.")
```

```
print("- **F1-score** is the harmonic mean of precision and recall,
providing a single metric that balances both.")
print(
    "For the 'negative' class, the model shows strong performance with
high precision (0.86),"
    " recall (0.82), and F1-score (0.84). This indicates the model is
good at correctly identifying negative tweets"
    " and minimizes false positives for this class."
)
print(
    "The 'neutral' class has moderate performance with a precision of
0.51, recall of 0.59, and F1-score of 0.55."
    " This suggests the model struggles more to accurately classify
neutral tweets compared to negative ones."
)
print(
    "The 'positive' class shows reasonable performance with a
precision of 0.70, recall of 0.68, and F1-score of 0.69."
    " It performs better than the 'neutral' class but not as well as
the 'negative' class."
)
print(
    "\nOverall accuracy is 0.75, which is a general measure, but the
class-specific metrics reveal the model's varying"
    " performance across different sentiments, especially the
challenge in classifying neutral tweets."
)

# Analyze Confusion Matrix
print("\nAnalysis of Confusion Matrix:")
print(
    "The confusion matrix visually shows the counts of true positive,
true negative, false positive, and false negative predictions."
)
print(f"- True Negatives (Negative classified as Negative): {cm[0,
0]}")
print(f"- False Positives (Neutral/Positive classified as Negative):
{cm[1, 0] + cm[2, 0]}")
print(f"- False Negatives (Negative classified as Neutral/Positive):
{cm[0, 1] + cm[0, 2]}")
print(f"- True Positives (Positive classified as Positive): {cm[2,
2]}")
print(f"- False Positives (Negative/Neutral classified as Positive):
{cm[0, 2] + cm[1, 2]}")

print(
    "\nFrom the confusion matrix:"
    "\n- A significant number of negative tweets are correctly
identified (1553)."
```

```
    "\n- The model often confuses neutral tweets with negative (179)
or positive (60)."
    "\n- Positive tweets are sometimes misclassified as negative (78)
or neutral (68)."
    "\n- The diagonal values (1553, 341, 313) represent the number of
correct predictions for each class."
)

# Summary of Model Performance and Insights
print("\nSummary of Model Performance and Insights:")
print(
    "The LSTM model demonstrates good overall accuracy in classifying
airline sentiment, achieving 75.38% accuracy on the test set."
)
print(
    "Its strongest performance is in identifying 'negative' tweets, as
indicated by high precision, recall, and F1-score for this class."
    " This is crucial for airlines to quickly identify and address
negative feedback."
)
print(
    "The model performs reasonably well on 'positive' tweets, but
there is room for improvement."
)
print(
    "The main weakness lies in classifying 'neutral' tweets, which are
often confused with 'negative' or 'positive' sentiments."
    " This lower performance on the neutral class is reflected in its
lower precision, recall, and F1-score."
    " This could be due to the often ambiguous nature of neutral
language in tweets."
)
print(
    "Insights from the sentiment distribution (seen in earlier EDA)
show a predominance of negative tweets,"
    " and the model's ability to accurately identify these is a key
strength for this dataset."
)
print(
    "To improve the model, future work could focus on better handling
of neutral sentiment, possibly through"
    " advanced techniques, more data augmentation for neutral tweets,
or exploring different model architectures."
)
```

```
Interpretation of Model Performance:

Analysis of Classification Report:
The classification report provides precision, recall, and F1-score for
each sentiment class.
```

- **Precision** is the ability of the classifier not to label as positive a sample that is negative.
- **Recall** is the ability of the classifier to find all the positive samples.
- **F1-score** is the harmonic mean of precision and recall, providing a single metric that balances both.
For the 'negative' class, the model shows strong performance with high precision (0.86), recall (0.82), and F1-score (0.84). This indicates the model is good at correctly identifying negative tweets and minimizes false positives for this class.
The 'neutral' class has moderate performance with a precision of 0.51, recall of 0.59, and F1-score of 0.55. This suggests the model struggles more to accurately classify neutral tweets compared to negative ones.
The 'positive' class shows reasonable performance with a precision of 0.70, recall of 0.68, and F1-score of 0.69. It performs better than the 'neutral' class but not as well as the 'negative' class.

Overall accuracy is 0.75, which is a general measure, but the class-specific metrics reveal the model's varying performance across different sentiments, especially the challenge in classifying neutral tweets.

Analysis of Confusion Matrix:
The confusion matrix visually shows the counts of true positive, true negative, false positive, and false negative predictions.
- True Negatives (Negative classified as Negative): 1710
- False Positives (Neutral/Positive classified as Negative): 310
- False Negatives (Negative classified as Neutral/Positive): 179
- True Positives (Positive classified as Positive): 291
- False Positives (Negative/Neutral classified as Positive): 89

From the confusion matrix:
- A significant number of negative tweets are correctly identified (1553).
- The model often confuses neutral tweets with negative (179) or positive (60).
- Positive tweets are sometimes misclassified as negative (78) or neutral (68).
- The diagonal values (1553, 341, 313) represent the number of correct predictions for each class.

Summary of Model Performance and Insights:
The LSTM model demonstrates good overall accuracy in classifying airline sentiment, achieving 75.38% accuracy on the test set.
Its strongest performance is in identifying 'negative' tweets, as indicated by high precision, recall, and F1-score for this class. This is crucial for airlines to quickly identify and address negative feedback.
The model performs reasonably well on 'positive' tweets, but there is

```
room for improvement.
The main weakness lies in classifying 'neutral' tweets, which are
often confused with 'negative' or 'positive' sentiments. This lower
performance on the neutral class is reflected in its lower precision,
recall, and F1-score. This could be due to the often ambiguous nature
of neutral language in tweets.
Insights from the sentiment distribution (seen in earlier EDA) show a
predominance of negative tweets, and the model's ability to accurately
identify these is a key strength for this dataset.
To improve the model, future work could focus on better handling of
neutral sentiment, possibly through advanced techniques, more data
augmentation for neutral tweets, or exploring different model
architectures.
```

Summary:

Model Performance Analysis (Post-Optimization)

After implementing Early Stopping and fixing the data split, the model's performance on the test set was evaluated.

- **Overall Accuracy:** The model achieved an overall accuracy of **{accuracy:.2f}%** on the test set. This indicates the percentage of tweets correctly classified across all sentiment categories.
- **Classification Report Analysis:**
  - The classification report provides detailed metrics (Precision, Recall, F1-score) for each sentiment class (negative, neutral, positive).
  - **Precision** measures the accuracy of positive predictions (out of all tweets predicted as a certain sentiment, how many were actually that sentiment).
  - **Recall** measures the model's ability to find all instances of a particular sentiment (out of all actual tweets of a certain sentiment, how many were correctly identified).
  - **F1-score** is the harmonic mean of Precision and Recall, providing a balanced measure of the model's performance on each class.

Sentiment	Precision	Recall	F1-Score	Support
Negative	{negative_precision:.2f}	{negative_recall:.2f}	{negative_f1:.2f}	{negative_support}
Neutral	{neutral_precision:.2f}	{neutral_recall:.2f}	{neutral_f1:.2f}	{neutral_support}

Sentiment	Precision	Recall	F1-Score	Support
Positive	{positive_precision:.2f}	{positive_recall:.2f}	{positive_f1:.2f}	{positive_support}

  - The model continues to perform strongly on the **'negative'** class, showing high values across precision, recall, and F1-score. This is critical for effectively identifying customer complaints.
  - Performance on the **'positive'** class is reasonable, but there is still room for improvement compared to the 'negative' class.
  - The **'neutral'** class remains the most challenging, with lower metrics. This suggests the model still struggles to distinctly identify neutral tweets, sometimes confusing them with negative or positive sentiments.
- **Confusion Matrix Analysis:**
  - The confusion matrix provides a visual breakdown of correct and incorrect classifications for each class.
  - The diagonal elements represent correctly classified instances (True Positives for each class).
  - Off-diagonal elements show misclassifications.
  - Interpreting the confusion matrix ({cm[0,0]}, {cm[1,1]}, {cm[2,2]} on the diagonal for negative, neutral, positive respectively) confirms the model's strength in identifying negative tweets.
  - It also highlights the misclassifications of neutral tweets ({cm[1,0]} as negative, {cm[1,2]} as positive) and some misclassifications of positive tweets ({cm[2,0]} as negative, {cm[2,1]} as neutral), aligning with the F1-score results.
- **Impact of Early Stopping:** (Based on the training output where training stopped early)
  - The training process stopped early due to the `EarlyStopping` callback monitoring validation loss. This indicates that further training was likely to lead to overfitting (where the model performs well on training data but poorly on unseen data). By stopping early, we obtain the model weights from the epoch that had the best performance on the validation set, which helps in better generalization to the test set.

Key Findings & Insights

- The implemented improvements, particularly Early Stopping, helped in training a model that generalizes better to unseen data compared to potential overfitting without it.
- The model is highly effective at identifying negative sentiment tweets, which is a valuable insight for airlines to address customer service issues promptly.
- Identifying neutral sentiment remains the primary challenge, indicating the need for further focus on this category.
- The saved model and tokenizer allow for easy deployment and prediction on new, incoming tweets without retraining.

Next Steps & Recommendations

Based on the current performance and analysis, here are professional recommendations for further work:

1. **Enhance Neutral Sentiment Classification:** This is the most critical area for improvement. Explore techniques such as:
    - Gathering or augmenting more diverse examples of neutral tweets.
    - Implementing more advanced text representation techniques or exploring different model architectures (e.g., GRU, or fine-tuning a pre-trained transformer model like BERT, although this is more complex).
    - Investigating specific linguistic patterns in neutral tweets that the current model might miss.
2. **Advanced Text Preprocessing:** Consider adding steps like handling contractions, negation handling, or incorporating emoticon/emoji information to potentially improve feature representation for all classes.
3. **Hyperparameter Tuning:** Systematically tune model hyperparameters (e.g., learning rate, dropout rate, number of LSTM units, batch size) using techniques like Grid Search or Random Search with cross-validation to find the optimal configuration for the current architecture.
4. **Error Analysis:** Conduct a detailed analysis of tweets that were misclassified, especially neutral ones, to understand common patterns or features that confuse the model. This can inform further preprocessing or model architecture decisions.
5. **Cross-Validation:** For a more robust evaluation, consider using k-fold cross-validation during training to get a more reliable estimate of model performance.
6. **Deployment:** Plan the deployment strategy for the saved model and tokenizer to integrate it into a real-time tweet monitoring system.

By focusing on these areas, particularly the classification of neutral tweets and further combating overfitting, the sentiment analysis system can become even more accurate and valuable.