MELD Dataset Exploratory Data Analysis for Emotion Recognition

Sentiment and emotion analysis for customer service chatbot conversations

1. Import Required Libraries

```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)
```

```
Mon Aug  4 13:22:23 2025
+-----------------------------------------------------------------------------
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version:
|-----------------------------------------+------------------------+-----------
```

```
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Unc
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Co
|                                         |                      |
|=========================================+======================+============
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |
| N/A   47C    P8              9W /   70W |      0MiB /  15360MiB |      0%
|                                         |                      |
+-----------------------------------------+----------------------+------------

+-----------------------------------------------------------------------------
| Processes:
|  GPU   GI   CI        PID   Type   Process name                            GP
|        ID   ID                                                             Us
|=============================================================================
|  No running processes found
+-----------------------------------------------------------------------------
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import zipfile
from collections import Counter
import warnings
warnings.filterwarnings('ignore')
```

```
# Set style for better visualizations
plt.style.use('seaborn-v0_8-darkgrid')
sns.set_palette("husl")
```

2. Download and Extract MELD Dataset

```
'''
if not os.path.exists('data'):
    os.makedirs('data')
# Download MELD dataset if not already present
meld_url = "http://web.eecs.umich.edu/~mihalcea/downloads/MELD.Raw.tar.gz"
output_path = "data/MELD.Raw.tar.gz"

if not os.path.exists(output_path):
    print("Downloading MELD dataset...")
    wget.download(meld_url, output_path)
    print("\nDownload completed!")
else:
    print("Dataset already downloaded.")

# Extract the dataset
import tarfile
if not os.path.exists('data/MELD.Raw'):
```

```
    print("Extracting dataset...")
    with tarfile.open(output_path, 'r:gz') as tar:
        tar.extractall('data/')
    print("Extraction completed!")
'''
```

```
'\nif not os.path.exists(\'data\'):\n    os.makedirs(\'data\')\n# Download MELD
dataset if not already present\nmeld_url = "http://web.eecs.umich.edu/~mihalcea/
downloads/MELD.Raw.tar.gz"\noutput_path = "data/MELD.Raw.tar.gz"\n\nif not os.pa
th.exists(output_path):\n    print("Downloading MELD dataset...")\n    wget.down
load(meld_url, output_path)\n    print("\nDownload completed!")\nelse:\n    prin
t("Dataset already downloaded.")\n\n# Extract the dataset\nimport tarfile\nif no
```

3. Load Dataset Files

```
# Define paths to the CSV files
# Use raw GitHub URLs for direct file access
train_path = 'https://raw.githubusercontent.com/declare-lab/MELD/master/data/MELD/tra
dev_path = 'https://raw.githubusercontent.com/declare-lab/MELD/master/data/MELD/dev_s
test_path = 'https://raw.githubusercontent.com/declare-lab/MELD/master/data/MELD/test

# Load the datasets
print("Loading datasets...")
train_df = pd.read_csv(train_path)
dev_df = pd.read_csv(dev_path)
```

```
test_df = pd.read_csv(test_path)

print(f"Train set loaded: {train_df.shape}")
print(f"Dev set loaded: {dev_df.shape}")
print(f"Test set loaded: {test_df.shape}")
```

```
Loading datasets...
Train set loaded: (9989, 11)
Dev set loaded: (1109, 11)
Test set loaded: (2610, 11)
```

4. Initial Data Exploration

```
# Display basic information about the training dataset
print("=== Training Dataset Info ===")
print(train_df.info())
print("\n=== First 5 rows of training data ===")
print(train_df.head())

# Check column names
print("\n=== Column Names ===")
print(train_df.columns.tolist())
```

```
=== Training Dataset Info ===
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 9989 entries, 0 to 9988
Data columns (total 11 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   Sr No.        9989 non-null   int64 
 1   Utterance     9989 non-null   object
 2   Speaker       9989 non-null   object
 3   Emotion       9989 non-null   object
 4   Sentiment     9989 non-null   object
 5   Dialogue_ID   9989 non-null   int64 
 6   Utterance_ID  9989 non-null   int64 
 7   Season        9989 non-null   int64 
 8   Episode       9989 non-null   int64 
 9   StartTime     9989 non-null   object
 10  EndTime       9989 non-null   object
dtypes: int64(5), object(6)
memory usage: 858.6+ KB
None

=== First 5 rows of training data ===
   Sr No.                                          Utterance          Speaker  \
0       1  also I was the point person on my company’s tr...         Chandler   
1       2                   You must’ve had your hands full.  The Interviewer   
2       3                            That I did. That I did.         Chandler   
3       4      So let’s talk a little bit about your duties.  The Interviewer   
4       5                             My duties?  All right.         Chandler   

    Emotion Sentiment  Dialogue_ID  Utterance_ID  Season  Episode  \
0   neutral   neutral            0             0       8       21   
```

```
1   neutral   neutral            0             1       8       21
2   neutral   neutral            0             2       8       21
3   neutral   neutral            0             3       8       21
4  surprise  positive            0             4       8       21

     StartTime       EndTime
0  00:16:16,059  00:16:21,731
1  00:16:21,940  00:16:23,442
2  00:16:23,442  00:16:26,389
3  00:16:26,820  00:16:29,572
4  00:16:34,452  00:16:40,917

=== Column Names ===
['Sr No.', 'Utterance', 'Speaker', 'Emotion', 'Sentiment', 'Dialogue_ID', 'Uttera
```

```
# Display basic information about the training dataset
print("=== Training Dataset Info ===")
print(train_df.info())
print("\n=== First 5 rows of training data ===")
print(train_df.head())

# Check column names
print("\n=== Column Names ===")
print(train_df.columns.tolist())
```

```
=== Training Dataset Info ===
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 9989 entries, 0 to 9988
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   Sr No.         9989 non-null   int64 
 1   Utterance      9989 non-null   object
 2   Speaker        9989 non-null   object
 3   Emotion        9989 non-null   object
 4   Sentiment      9989 non-null   object
 5   Dialogue_ID    9989 non-null   int64 
 6   Utterance_ID   9989 non-null   int64 
 7   Season         9989 non-null   int64 
 8   Episode        9989 non-null   int64 
 9   StartTime      9989 non-null   object
 10  EndTime        9989 non-null   object
dtypes: int64(5), object(6)
memory usage: 858.6+ KB
None

=== First 5 rows of training data ===
   Sr No.                                          Utterance          Speaker  \
0       1  also I was the point person on my company’s tr...         Chandler   
1       2                   You must’ve had your hands full.  The Interviewer   
2       3                            That I did. That I did.         Chandler   
3       4      So let’s talk a little bit about your duties.  The Interviewer   
4       5                             My duties?  All right.         Chandler   

    Emotion Sentiment  Dialogue_ID  Utterance_ID  Season  Episode  \
0   neutral   neutral            0             0       8       21   
```

```
1   neutral   neutral            0             1       8       21
2   neutral   neutral            0             2       8       21
3   neutral   neutral            0             3       8       21
4  surprise  positive            0             4       8       21

      StartTime       EndTime
0  00:16:16,059  00:16:21,731
1  00:16:21,940  00:16:23,442
2  00:16:23,442  00:16:26,389
3  00:16:26,820  00:16:29,572
4  00:16:34,452  00:16:40,917

=== Column Names ===
['Sr No.', 'Utterance', 'Speaker', 'Emotion', 'Sentiment', 'Dialogue_ID', 'Uttera
```

5. Dataset Statistics

```
# Combined dataset statistics
total_utterances = len(train_df) + len(dev_df) + len(test_df)
print(f"Total utterances in dataset: {total_utterances}")
print(f"Training set: {len(train_df)} ({len(train_df)/total_utterances*100:.1f}%)")
print(f"Development set: {len(dev_df)} ({len(dev_df)/total_utterances*100:.1f}%)")
print(f"Test set: {len(test_df)} ({len(test_df)/total_utterances*100:.1f}%)")

# Check for missing values
```

```
print("\n=== Missing Values in Training Set ===")
print(train_df.isnull().sum())
```

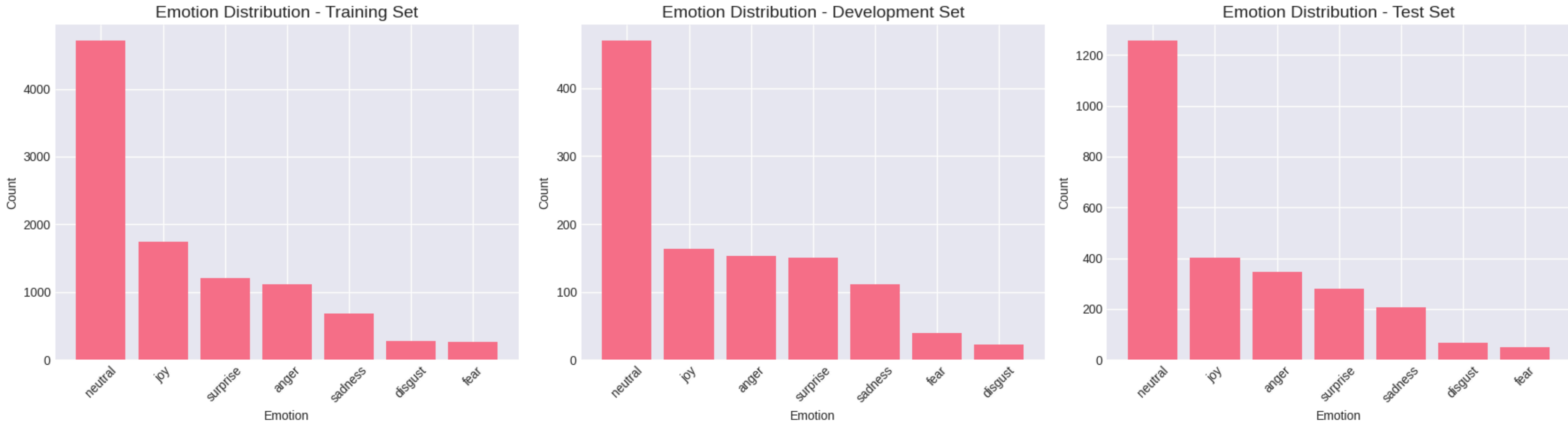
```
Total utterances in dataset: 13708
Training set: 9989 (72.9%)
Development set: 1109 (8.1%)
Test set: 2610 (19.0%)

=== Missing Values in Training Set ===
Sr No.          0
Utterance       0
Speaker         0
Emotion         0
Sentiment       0
Dialogue_ID     0
Utterance_ID    0
Season          0
Episode         0
StartTime       0
EndTime         0
dtype: int64
```

6. Emotion Distribution Analysis

```
# Analyze emotion distribution across datasets
def plot_emotion_distribution(train_df, dev_df, test_df):
```

```python
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

# Training set
emotion_counts_train = train_df['Emotion'].value_counts()
axes[0].bar(emotion_counts_train.index, emotion_counts_train.values)
axes[0].set_title('Emotion Distribution - Training Set', fontsize=14)
axes[0].set_xlabel('Emotion')
axes[0].set_ylabel('Count')
axes[0].tick_params(axis='x', rotation=45)

# Dev set
emotion_counts_dev = dev_df['Emotion'].value_counts()
axes[1].bar(emotion_counts_dev.index, emotion_counts_dev.values)
axes[1].set_title('Emotion Distribution - Development Set', fontsize=14)
axes[1].set_xlabel('Emotion')
axes[1].set_ylabel('Count')
axes[1].tick_params(axis='x', rotation=45)

# Test set
emotion_counts_test = test_df['Emotion'].value_counts()
axes[2].bar(emotion_counts_test.index, emotion_counts_test.values)
axes[2].set_title('Emotion Distribution - Test Set', fontsize=14)
axes[2].set_xlabel('Emotion')
axes[2].set_ylabel('Count')
axes[2].tick_params(axis='x', rotation=45)

plt.tight_layout()
```

```
    plt.show()

plot_emotion_distribution(train_df, dev_df, test_df)
```




```
# Overall emotion distribution
all_emotions = pd.concat([train_df['Emotion'], dev_df['Emotion'], test_df['Emotion']]
```

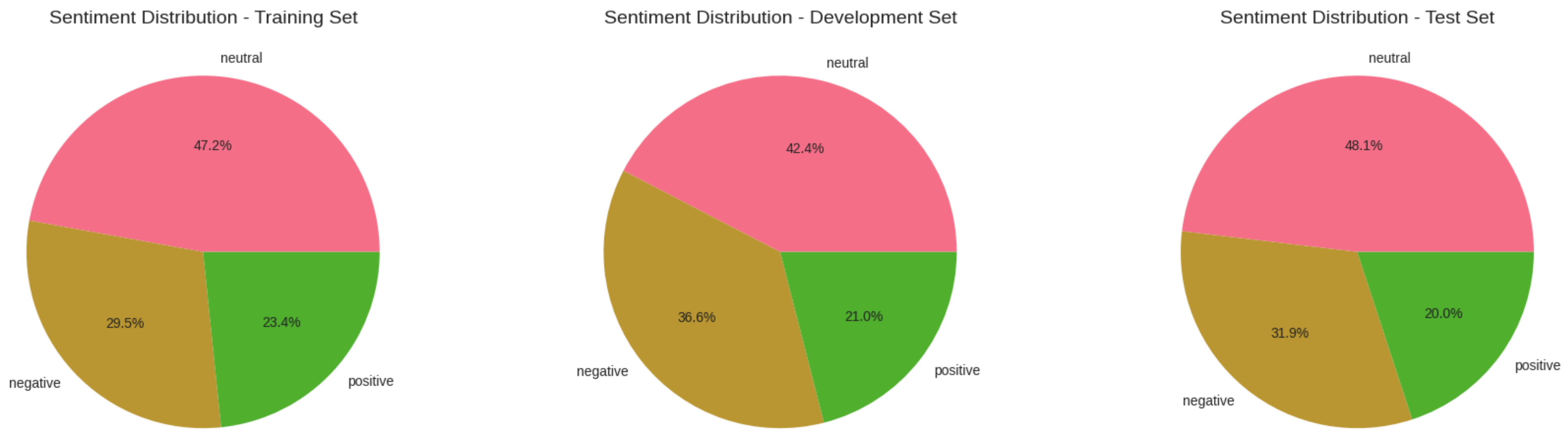
```
emotion_dist = all_emotions.value_counts()
print("\n=== Overall Emotion Distribution ===")
print(emotion_dist)
print(f"\nTotal unique emotions: {len(emotion_dist)}")
```

```
=== Overall Emotion Distribution ===
Emotion
neutral     6436
joy         2308
surprise    1636
anger       1607
sadness     1002
disgust      361
fear         358
Name: count, dtype: int64

Total unique emotions: 7
```

7. Sentiment Distribution Analysis

```
# Analyze sentiment distribution
def plot_sentiment_distribution(train_df, dev_df, test_df):
    fig, axes = plt.subplots(1, 3, figsize=(18, 5))
```

```python
    datasets = [('Training', train_df), ('Development', dev_df), ('Test', test_df)]

    for idx, (name, df) in enumerate(datasets):
        sentiment_counts = df['Sentiment'].value_counts()
        axes[idx].pie(sentiment_counts.values, labels=sentiment_counts.index, autopct
        axes[idx].set_title(f'Sentiment Distribution - {name} Set', fontsize=14)

    plt.tight_layout()
    plt.show()

plot_sentiment_distribution(train_df, dev_df, test_df)
```




```
# Overall sentiment distribution
all_sentiments = pd.concat([train_df['Sentiment'], dev_df['Sentiment'], test_df['Sent
sentiment_dist = all_sentiments.value_counts()
```

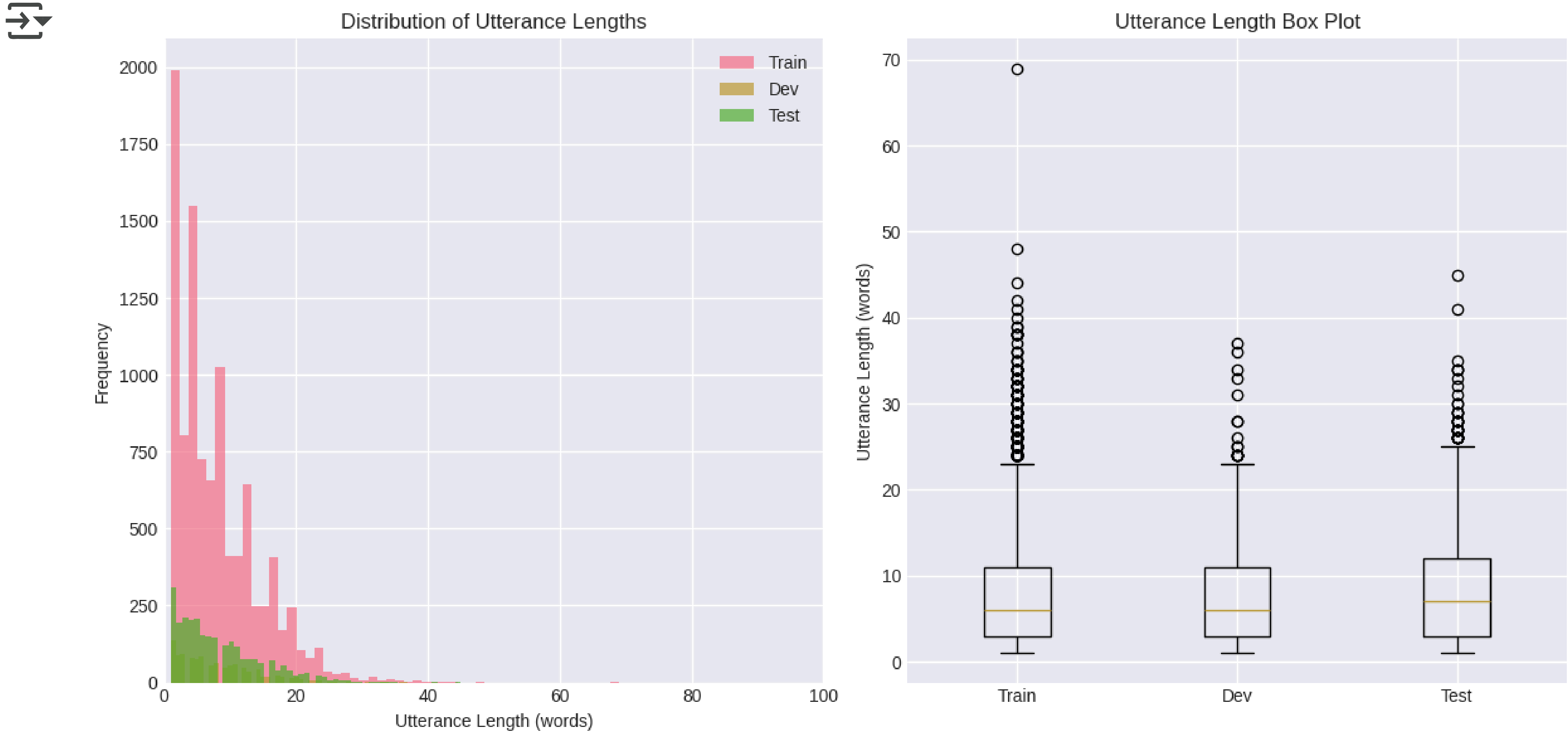
```
print("\n=== Overall Sentiment Distribution ===")
print(sentiment_dist)
```

```
=== Overall Sentiment Distribution ===
Sentiment
neutral     6436
negative    4184
positive    3088
Name: count, dtype: int64
```

8. Text Length Analysis

```
# Analyze utterance lengths
train_df['utterance_length'] = train_df['Utterance'].apply(lambda x: len(str(x).split
dev_df['utterance_length'] = dev_df['Utterance'].apply(lambda x: len(str(x).split()))
test_df['utterance_length'] = test_df['Utterance'].apply(lambda x: len(str(x).split()

# Plot utterance length distribution
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.hist(train_df['utterance_length'], bins=50, alpha=0.7, label='Train')
plt.hist(dev_df['utterance_length'], bins=50, alpha=0.7, label='Dev')
plt.hist(test_df['utterance_length'], bins=50, alpha=0.7, label='Test')
```

```
plt.xlabel('Utterance Length (words)')
plt.ylabel('Frequency')
plt.title('Distribution of Utterance Lengths')
plt.legend()
plt.xlim(0, 100)

plt.subplot(1, 2, 2)
plt.boxplot([train_df['utterance_length'], dev_df['utterance_length'], test_df['utter
            labels=['Train', 'Dev', 'Test'])
plt.ylabel('Utterance Length (words)')
plt.title('Utterance Length Box Plot')
plt.tight_layout()
plt.show()
```


Distribution of Utterance Lengths
Train
Dev
Test
Frequency
Utterance Length (words)
0
250
500
750
1000
1250
1500
1750
2000
0
20
40
60
80
100
Utterance Length Box Plot
Utterance Length (words)
0
10
20
30
40
50
60
70
Train
Dev
Test

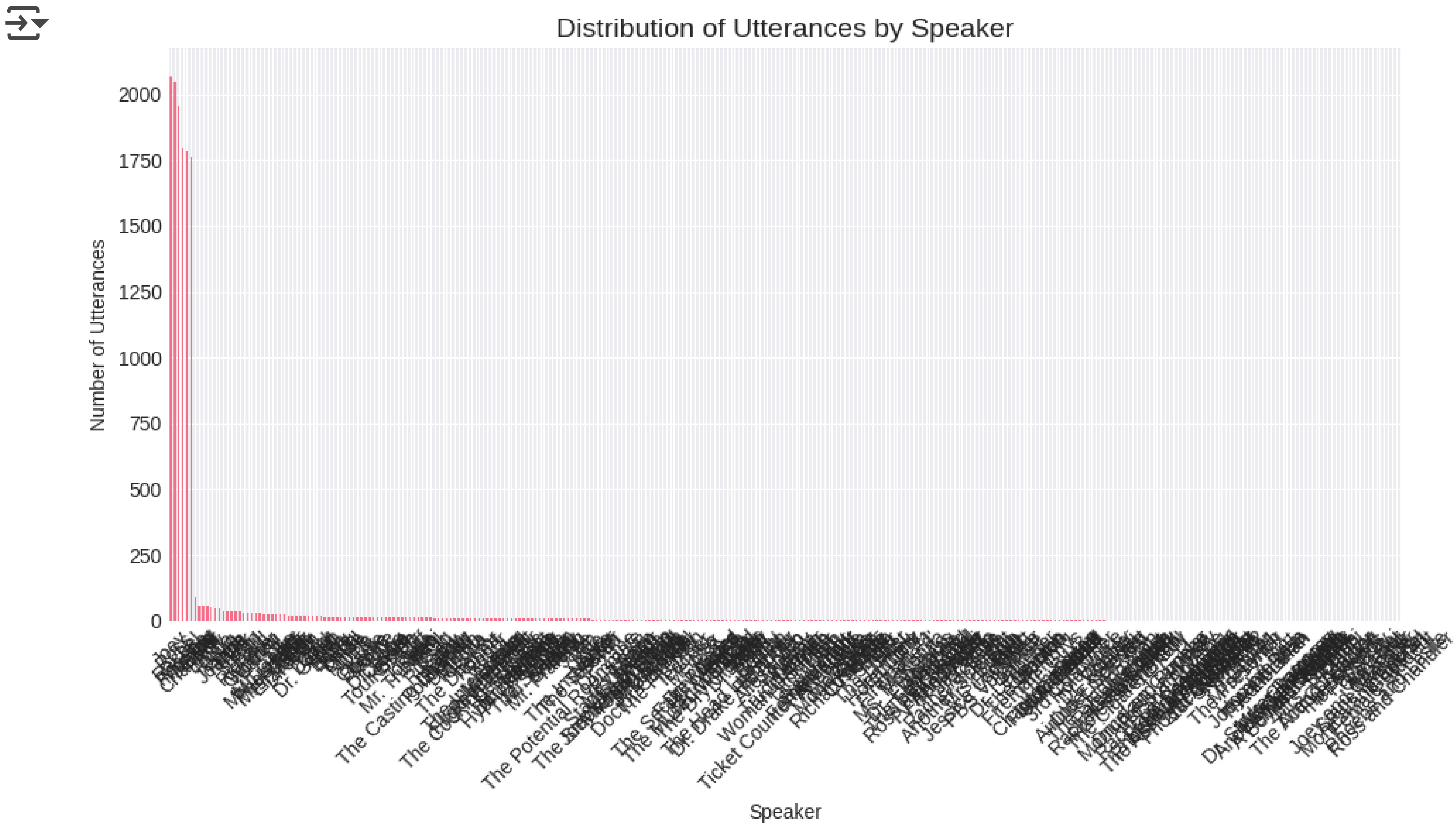
```
# Statistics
print("=== Utterance Length Statistics ===")
print(f"Training set - Mean: {train_df['utterance_length'].mean():.2f}, Median: {trai
print(f"Dev set - Mean: {dev_df['utterance_length'].mean():.2f}, Median: {dev_df['utt
print(f"Test set - Mean: {test_df['utterance_length'].mean():.2f}, Median: {test_df['
```

```
=== Utterance Length Statistics ===
Training set - Mean: 7.95, Median: 6, Max: 69
Dev set - Mean: 7.91, Median: 6, Max: 37
Test set - Mean: 8.21, Median: 7, Max: 45
```

9. Speaker Analysis

```
# Analyze speaker distribution
all_speakers = pd.concat([train_df['Speaker'], dev_df['Speaker'], test_df['Speaker']]
speaker_counts = all_speakers.value_counts()

plt.figure(figsize=(10, 6))
```

```
speaker_counts.plot(kind='bar')
plt.title('Distribution of Utterances by Speaker', fontsize=14)
plt.xlabel('Speaker')
plt.ylabel('Number of Utterances')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

print(f"Total unique speakers: {len(speaker_counts)}")
print("\n=== Speaker Distribution ===")
print(speaker_counts)
```



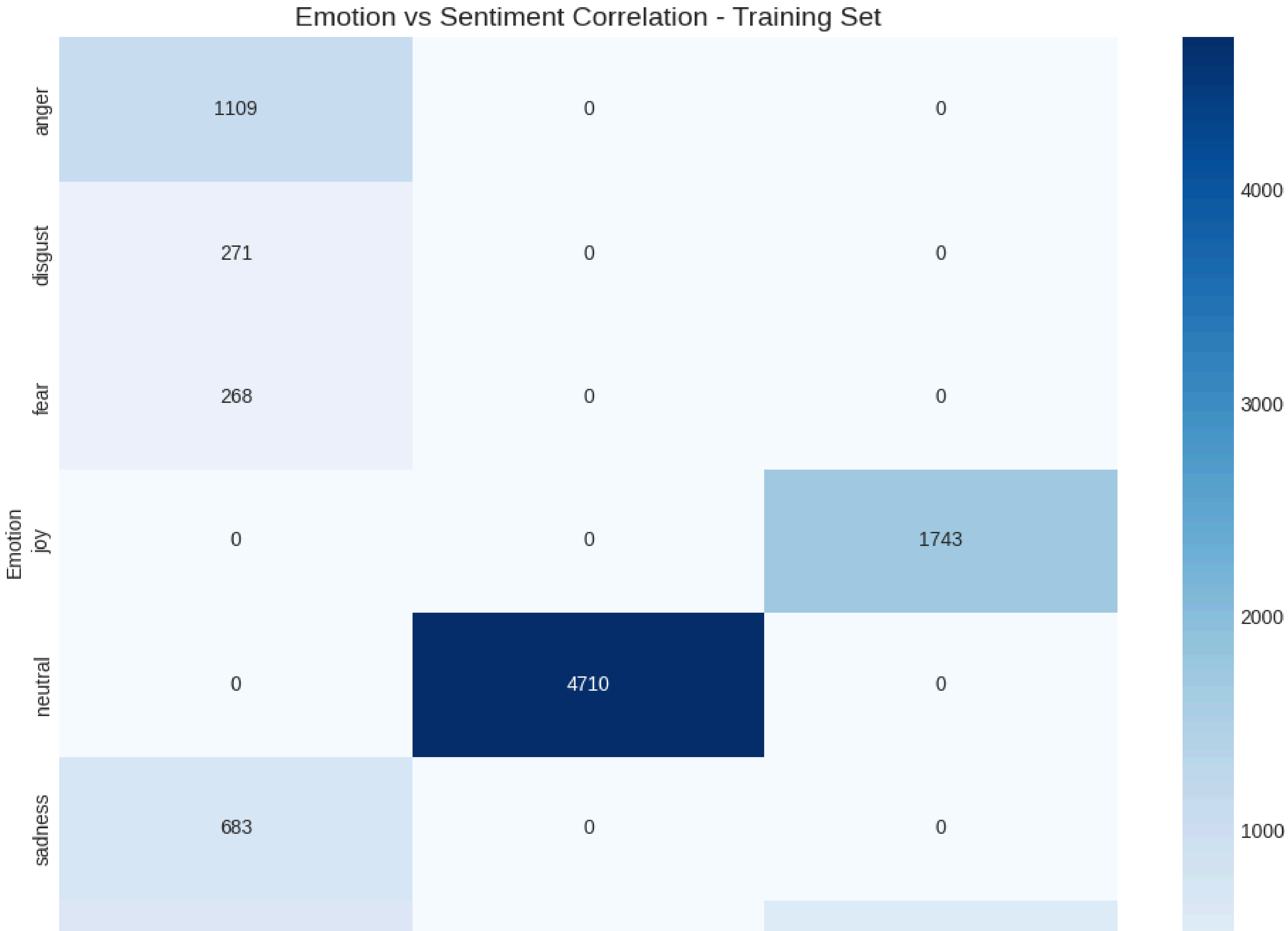

```
Total unique speakers: 304

=== Speaker Distribution ===
Speaker
Joey                     2070
Ross                     2048
```

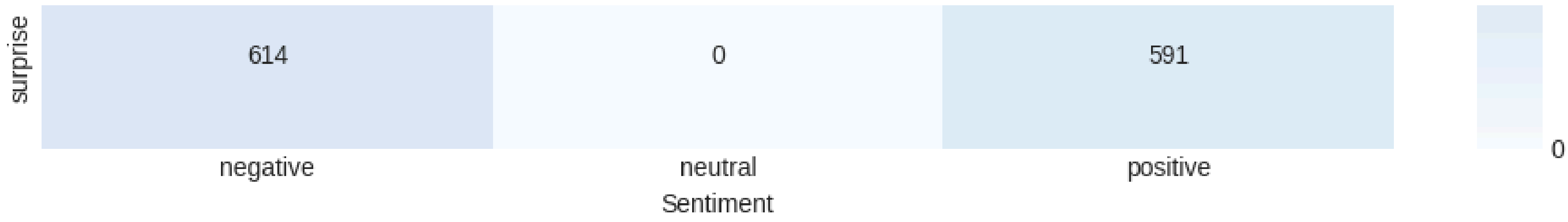
```
Rachel                 1955
Phoebe                 1797
Monica                 1782
                       ...
Female Student            1
Phoebe and Leslie         1
Ross and Chandler         1
Frank Sr.                 1
Guest #1                  1
Name: count, Length: 304, dtype: int64
```

10. Emotion-Sentiment Correlation

```
# Analyze correlation between emotion and sentiment
def create_emotion_sentiment_heatmap(df, title):
    emotion_sentiment_crosstab = pd.crosstab(df['Emotion'], df['Sentiment'])

    plt.figure(figsize=(10, 8))
    sns.heatmap(emotion_sentiment_crosstab, annot=True, fmt='d', cmap='Blues')
    plt.title(f'Emotion vs Sentiment Correlation - {title}', fontsize=14)
    plt.xlabel('Sentiment')
    plt.ylabel('Emotion')
    plt.tight_layout()
    plt.show()

    return emotion_sentiment_crosstab

print("=== Training Set Emotion-Sentiment Correlation ===")
train_crosstab = create_emotion_sentiment_heatmap(train_df, 'Training Set')
```

=== Training Set Emotion-Sentiment Correlation ===

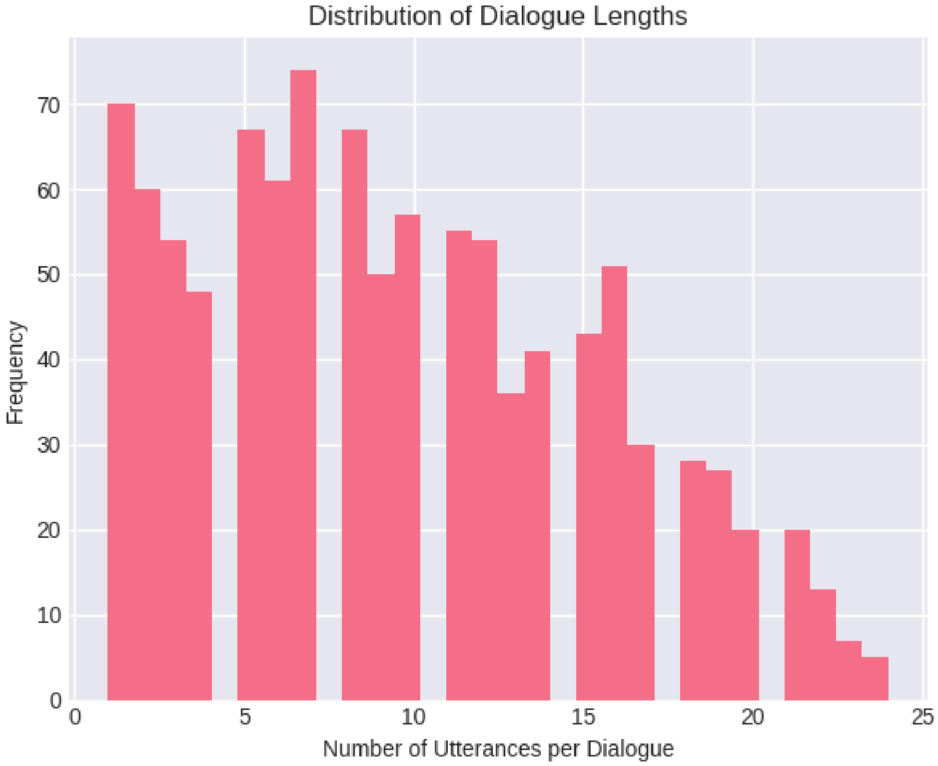


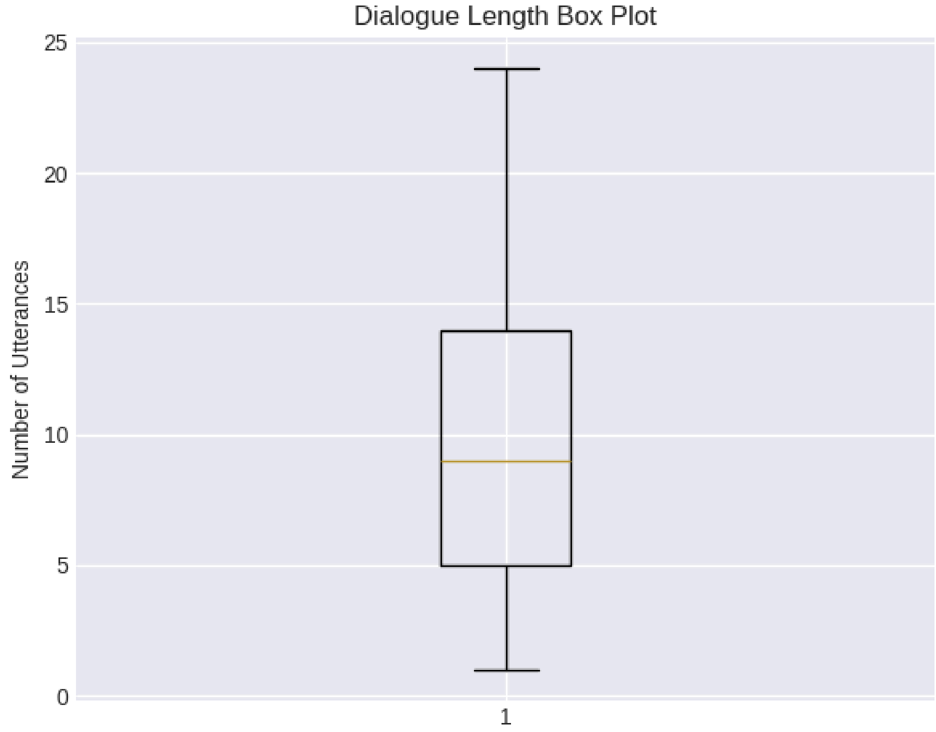

surprise
614
0
591
negative
neutral
positive
Sentiment
0

11. Dialogue Context Analysis

```
# Analyze dialogue structure
dialogue_counts = train_df['Dialogue_ID'].value_counts()
print(f"Total dialogues in training set: {len(dialogue_counts)}")
print(f"Average utterances per dialogue: {dialogue_counts.mean():.2f}")
print(f"Min utterances in a dialogue: {dialogue_counts.min()}")
print(f"Max utterances in a dialogue: {dialogue_counts.max()}")

# Plot dialogue length distribution
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.hist(dialogue_counts.values, bins=30)
plt.xlabel('Number of Utterances per Dialogue')
plt.ylabel('Frequency')
plt.title('Distribution of Dialogue Lengths')

plt.subplot(1, 2, 2)
plt.boxplot(dialogue_counts.values)
plt.ylabel('Number of Utterances')
plt.title('Dialogue Length Box Plot')
plt.tight_layout()
plt.show()
```

```
Total dialogues in training set: 1038
Average utterances per dialogue: 9.62
Min utterances in a dialogue: 1
Max utterances in a dialogue: 24
```





12. Sample Conversations Analysis

```
# Display sample conversations with emotions
def display_sample_dialogue(df, dialogue_id, max_utterances=10):
    dialogue = df[df['Dialogue_ID'] == dialogue_id].head(max_utterances)
    print(f"\n=== Sample Dialogue (ID: {dialogue_id}) ===")
    for idx, row in dialogue.iterrows():
        print(f"{row['Speaker']}: {row['Utterance']}")
        print(f"   Emotion: {row['Emotion']}, Sentiment: {row['Sentiment']}")
        print()

# Display a few sample dialogues
sample_dialogue_ids = train_df['Dialogue_ID'].unique()[:3]
for dialogue_id in sample_dialogue_ids:
    display_sample_dialogue(train_df, dialogue_id)
```

```
   Emotion: joy, Sentiment: positive

Monica: Chris says they’re closing down the bar.
   Emotion: sadness, Sentiment: negative

Chandler: No way!
   Emotion: surprise, Sentiment: negative

Monica: Yeah, apparently they’re turning it into some kinda coffee place.
   Emotion: neutral, Sentiment: neutral

Chandler: Just coffee! Where are we gonna hang out now?
   Emotion: disgust, Sentiment: negative

Monica: Got me.
   Emotion: sadness, Sentiment: negative

Chandler: Can I get a beer.
   Emotion: neutral, Sentiment: neutral

Monica: Hey, did you pick a roommate?
   Emotion: neutral, Sentiment: neutral
```

13. Word Frequency Analysis

```python
from collections import Counter
import re

# Function to clean and tokenize text
def tokenize(text):
    # Convert to lowercase and remove punctuation
    text = str(text).lower()
    text = re.sub(r'[^\w\s]', '', text)
    return text.split()

# Get word frequencies for each emotion
emotion_words = {}
for emotion in train_df['Emotion'].unique():
    emotion_utterances = train_df[train_df['Emotion'] == emotion]['Utterance']
    all_words = []
    for utterance in emotion_utterances:
        all_words.extend(tokenize(utterance))
    emotion_words[emotion] = Counter(all_words).most_common(20)


# Display top words for each emotion
print("=== Top 10 Words per Emotion ===")
for emotion, word_counts in emotion_words.items():
    print(f"\n{emotion}:")
    for word, count in word_counts[:10]:
        print(f"  {word}: {count}")
```

```
i: 132
you: 82
```

```
  oh: 232
  and: 223
  it: 216
  so: 164
  hey: 143

disgust:
  you: 96
  i: 78
  the: 74
  a: 69
  to: 51
  and: 44
  that: 44
  oh: 41
  no: 35
  it: 33

anger:
  you: 431
  i: 374
  the: 247
  to: 218
  and: 174
  a: 168
  it: 157
  that: 143
  me: 137
  no: 130
```

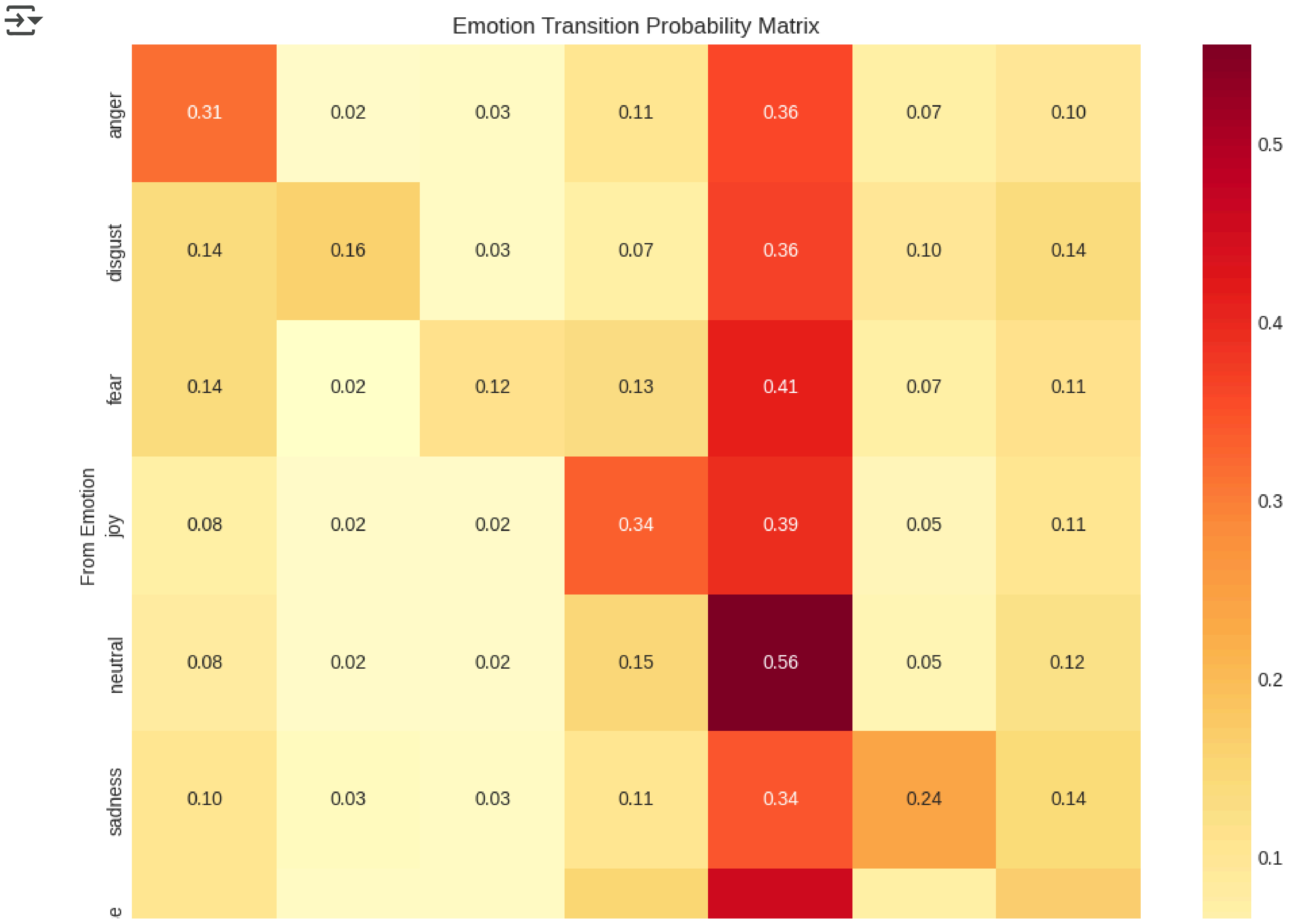
14. Emotion Transition Analysis

```
# Analyze emotion transitions within dialogues
def analyze_emotion_transitions(df):
    transitions = []

    for dialogue_id in df['Dialogue_ID'].unique():
        dialogue = df[df['Dialogue_ID'] == dialogue_id].sort_values('Utterance_ID')
        emotions = dialogue['Emotion'].tolist()

        for i in range(len(emotions) - 1):
            transitions.append((emotions[i], emotions[i+1]))

    return Counter(transitions)


# Get emotion transitions
transitions = analyze_emotion_transitions(train_df)
top_transitions = transitions.most_common(15)

print("=== Top 15 Emotion Transitions ===")
for (from_emotion, to_emotion), count in top_transitions:
    print(f"{from_emotion} → {to_emotion}: {count}")
```

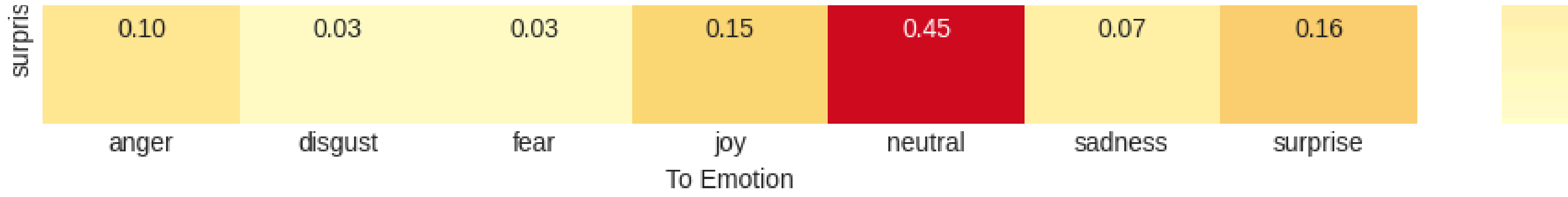
```
=== Top 15 Emotion Transitions ===
neutral → neutral: 2354
neutral → joy: 631
joy → neutral: 602
joy → joy: 520
neutral → surprise: 500
surprise → neutral: 492
anger → neutral: 358
neutral → anger: 352
anger → anger: 314
neutral → sadness: 212
sadness → neutral: 206
surprise → surprise: 179
surprise → joy: 167
joy → surprise: 165
sadness → sadness: 147
```

```python
# Create transition matrix
unique_emotions = sorted(train_df['Emotion'].unique())
transition_matrix = pd.DataFrame(0, index=unique_emotions, columns=unique_emotions)

for (from_emotion, to_emotion), count in transitions.items():
    transition_matrix.loc[from_emotion, to_emotion] = count


# Normalize by row to get probabilities
transition_prob = transition_matrix.div(transition_matrix.sum(axis=1), axis=0)
```

```
plt.figure(figsize=(10, 8))
sns.heatmap(transition_prob, annot=True, fmt='.2f', cmap='YlOrRd')
plt.title('Emotion Transition Probability Matrix')
plt.xlabel('To Emotion')
plt.ylabel('From Emotion')
plt.tight_layout()
plt.show()
```



Emotion Transition Probability Matrix
From Emotion
anger
disgust
fear
joy
neutral
sadness
0.31
0.02
0.03
0.11
0.36
0.07
0.10
0.14
0.16
0.03
0.07
0.36
0.10
0.14
0.14
0.02
0.12
0.13
0.41
0.07
0.11
0.08
0.02
0.02
0.34
0.39
0.05
0.11
0.08
0.02
0.02
0.15
0.56
0.05
0.12
0.10
0.03
0.03
0.11
0.34
0.24
0.14
0.5
0.4
0.3
0.2
0.1


surpris
0.10
0.03
0.03
0.15
0.45
0.07
0.16
anger
disgust
fear
joy
neutral
sadness
surprise
To Emotion

15. Summary Statistics and Insights

```
print("=== MELD Dataset Summary ===")
print(f"Total utterances: {total_utterances}")
print(f"Total unique dialogues: {len(pd.concat([train_df['Dialogue_ID'], dev_df['Dial
print(f"Average utterance length: {pd.concat([train_df['utterance_length'], dev_df['u
print(f"\nEmotion classes: {sorted(train_df['Emotion'].unique())}")
print(f"Sentiment classes: {sorted(train_df['Sentiment'].unique())}")

# Class imbalance analysis
print("\n=== Class Imbalance Analysis ===")
emotion_imbalance = emotion_dist.max() / emotion_dist.min()
sentiment_imbalance = sentiment_dist.max() / sentiment_dist.min()
print(f"Emotion class imbalance ratio: {emotion_imbalance:.2f}")
print(f"Sentiment class imbalance ratio: {sentiment_imbalance:.2f}")
```

```
=== MELD Dataset Summary ===
Total utterances: 13708
Total unique dialogues: 1039
Average utterance length: 8.00 words

Emotion classes: ['anger', 'disgust', 'fear', 'joy', 'neutral', 'sadness', 'surpr
Sentiment classes: ['negative', 'neutral', 'positive']
```

```
=== Class Imbalance Analysis ===
Emotion class imbalance ratio: 17.98
Sentiment class imbalance ratio: 2.08
```

```
# Key insights for model development
print("\n=== Key Insights for Model Development ===")
print("1. The dataset shows significant class imbalance, especially in emotions")
print("2. Neutral emotion is dominant, which might affect model performance")
print("3. Average utterance length is relatively short, suitable for transformer mode
print("4. Strong correlation between certain emotions and sentiments")
print("5. Context from dialogue flow could be important for emotion recognition")
```

```
=== Key Insights for Model Development ===
1. The dataset shows significant class imbalance, especially in emotions
2. Neutral emotion is dominant, which might affect model performance
3. Average utterance length is relatively short, suitable for transformer models
4. Strong correlation between certain emotions and sentiments
5. Context from dialogue flow could be important for emotion recognition
```

16. Prepare Data for Transformer Models

```
# Create a function to prepare data for transformer models
def prepare_for_transformers(df):
```

```
"""
Prepare the dataset for transformer-based models
"""
# Create emotion to index mapping
emotion_to_idx = {emotion: idx for idx, emotion in enumerate(sorted(df['Emotion']
sentiment_to_idx = {sentiment: idx for idx, sentiment in enumerate(sorted(df['Ser

# Add numerical labels
df['emotion_label'] = df['Emotion'].map(emotion_to_idx)
df['sentiment_label'] = df['Sentiment'].map(sentiment_to_idx)

# Group by dialogue for context modeling
dialogues = []
for dialogue_id in df['Dialogue_ID'].unique():
    dialogue = df[df['Dialogue_ID'] == dialogue_id].sort_values('Utterance_ID')
    dialogues.append({
        'dialogue_id': dialogue_id,
        'utterances': dialogue['Utterance'].tolist(),
        'speakers': dialogue['Speaker'].tolist(),
        'emotions': dialogue['emotion_label'].tolist(),
        'sentiments': dialogue['sentiment_label'].tolist()
    })

return dialogues, emotion_to_idx, sentiment_to_idx
```

```
# Prepare training data
train_dialogues, emotion_to_idx, sentiment_to_idx = prepare_for_transformers(train_df
print(f"Prepared {len(train_dialogues)} dialogues for training")
print(f"\nEmotion mapping: {emotion_to_idx}")
print(f"\nSentiment mapping: {sentiment_to_idx}")
```

```
Prepared 1038 dialogues for training

Emotion mapping: {'anger': 0, 'disgust': 1, 'fear': 2, 'joy': 3, 'neutral': 4, 's

Sentiment mapping: {'negative': 0, 'neutral': 1, 'positive': 2}
```

```
# Save mappings for later use
import json
if not os.path.exists('data'):
    os.makedirs('data')

mappings = {
    'emotion_to_idx': emotion_to_idx,
    'sentiment_to_idx': sentiment_to_idx,
    'idx_to_emotion': {v: k for k, v in emotion_to_idx.items()},
    'idx_to_sentiment': {v: k for k, v in sentiment_to_idx.items()}
}

with open('data/label_mappings.json', 'w') as f:
```

```
    json.dump(mappings, f, indent=2)
print("\nLabel mappings saved to 'data/label_mappings.json'")
```

```
Label mappings saved to 'data/label_mappings.json'
```

Data PreProcessing Summary

This EDA reveals several important insights for building emotion recognition models:

1. **Class Imbalance**: The dataset shows significant imbalance, particularly with "neutral" emotion being dominant. Consider using weighted loss functions or resampling techniques.
2. **Multi-modal Nature**: While we focused on text, MELD includes audio and visual features that could enhance model performance.
3. **Context Importance**: Emotions often depend on dialogue context, making this dataset ideal for context-aware transformer models.
4. **Short Utterances**: Most utterances are relatively short (median ~10 words), which is suitable for transformer architectures.
5. **Speaker Patterns**: Different speakers show distinct emotion patterns, which could be leveraged in multi-speaker models.

Next steps:

- Implement transformer-based models (BERT, RoBERTa, etc.) for emotion classification
- Experiment with context-aware architectures that consider dialogue history
- Apply techniques to handle class imbalance
- Consider multi-task learning for joint emotion and sentiment prediction

Model Implementation

Implementation of LSTM, BERT, RoBERTa with context-aware models using TensorFlow

1. Import Required Libraries and Setup

[ ] ↳ 3 cells hidden

2. Model Performance Optimization and Fine Tuning

2.1 Model Enhancement

[ ] ↳ 4 cells hidden

2.2 Dataset Processing and Generator

[ ] ↳ 2 cells hidden

2.3 Loss Function

[ ] ↳ 2 cells hidden

2.4 Model Training Functions

[ ] ↳ 3 cells hidden

3. Model Training and Evaluation

3.1 `main()` function: Training BERT Model

```
import numpy as np
import pandas as pd
import tensorflow as tf
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from transformers import BertTokenizer, TFBertModel
from tensorflow.keras.layers import Input, Dense, Dropout, Embedding, LSTM, Bidirecti
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from sklearn.preprocessing import LabelEncoder
from sklearn.utils.class_weight import compute_class_weight
from imblearn.over_sampling import SMOTE
from sklearn.metrics import balanced_accuracy_score
# Usage

train_path = 'https://raw.githubusercontent.com/declare-lab/MELD/master/data/MELD/tra
dev_path = 'https://raw.githubusercontent.com/declare-lab/MELD/master/data/MELD/dev_s
test_path = 'https://raw.githubusercontent.com/declare-lab/MELD/master/data/MELD/test
if __name__ == "__main__":
    print("🚀 Starting Enhanced Model Training...")
    try:
        model_types = ['bert'] #['bert', 'lstm', 'dialoguernn', 'cosmic']
```

```
        for model_type in model_types:
            model, history, results, mappings = train_enhanced_model(model_type)
        print("✅ Enhanced training completed successfully!")
    except Exception as e:
        print(f"❌ Error: {str(e)}")
        import traceback
        traceback.print_exc()
```

```
🚀 Starting Enhanced Model Training...
Loading and preparing data...
Applying advanced data augmentation...
Starting data augmentation...
Original emotion distribution: [1109  271  268 1743 4710  683 1205]
Target count for minority classes: 1648
Augmenting emotion 0: +539 samples
Augmenting emotion 1: +1377 samples
Augmenting emotion 2: +1380 samples
Augmenting emotion 5: +965 samples
Augmenting emotion 6: +443 samples
Augmentation complete. New dataset size: 5742
Final training set size: 5742
```

tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 4.63kB/s]

config.json: 100% 570/570 [00:00<00:00, 63.9kB/s]

vocab.txt: 100% 232k/232k [00:00<00:00, 6.74MB/s]

tokenizer.json: 100% 466k/466k [00:00<00:00, 6.93MB/s]

```
Final emotion distribution: [1648 1648 1648 1743 4710 1648 1648]
Initializing enhanced bert model...
```

model.safetensors: 100% 440M/440M [00:09<00:00, 56.9MB/s]

```
TensorFlow and JAX classes are deprecated and will be removed in Transformers v5.
Some weights of the PyTorch model were not used when initializing the TF 2.0 mode
- This IS expected if you are initializing TFBertModel from a PyTorch model train
- This IS NOT expected if you are initializing TFBertModel from a PyTorch model t
All the weights of TFBertModel were initialized from the PyTorch model.
```

```
If your task is similar to the task the model of the checkpoint was trained on, y
TensorFlow and JAX classes are deprecated and will be removed in Transformers v5.
Enhanced model summary:
```

Model: "enhanced_bert_model"

Layer (type)	Output Shape	Param #
bidirectional (Bidirectional)	(8, 256, 384)	1,476,096
multi_head_attention (MultiHeadAttention)	(8, 1, 768)	1,772,544
layer_normalization (LayerNormalization)	(8, 256, 384)	768
layer_normalization_1 (LayerNormalization)	(8, 768)	1,536
dense (Dense)	(8, 768)	590,592
dropout (Dropout)	?	0
dense_1 (Dense)	(8, 768)	590,592
dropout_1 (Dropout)	?	0
dense_2 (Dense)	(8, 384)	295,296
dropout_2 (Dropout)	?	0
dense_3 (Dense)	(8, 192)	73,920

emotion (Dense)	(8, 7)	1,351
dense_4 (Dense)	(8, 384)	295,296
dropout_3 (Dropout)	?	0
dense_5 (Dense)	(8, 192)	73,920
sentiment (Dense)	(8, 3)	579

```
 Total params: 5,172,490 (19.73 MB)
 Trainable params: 5,172,490 (19.73 MB)
 Non-trainable params: 0 (0.00 B)
Starting enhanced training...

Epoch 1: LearningRateScheduler setting learning rate to 2e-05.
Epoch 1/10
1837/1837 ━━━━━━━━━━━━━━━━━━━━ 0s 199ms/step - emotion_accuracy: 0.3239 - emotion
Epoch 1: val_emotion_accuracy improved from -inf to 0.40397, saving model to best
1837/1837 ━━━━━━━━━━━━━━━━━━━━ 441s 224ms/step - emotion_accuracy: 0.3240 - emoti

Epoch 2: LearningRateScheduler setting learning rate to 2e-05.
Epoch 2/10
1837/1837 ━━━━━━━━━━━━━━━━━━━━ 0s 199ms/step - emotion_accuracy: 0.4731 - emotion
Epoch 2: val_emotion_accuracy improved from 0.40397 to 0.52209, saving model to b
1837/1837 ━━━━━━━━━━━━━━━━━━━━ 388s 211ms/step - emotion_accuracy: 0.4731 - emoti

Epoch 3: LearningRateScheduler setting learning rate to 1.9e-05.
Epoch 3/10
1837/1837 ━━━━━━━━━━━━━━━━━━━━ 0s 205ms/step - emotion_accuracy: 0.5553 - emotion
Epoch 3: val_emotion_accuracy improved from 0.52209 to 0.54283, saving model to b
```

```
Epoch 3: val_emotion_accuracy improved from 0.52209 to 0.54283, saving model to b
1837/1837 ━━━━━━━━━━━━━━━━━━━━ 398s 217ms/step - emotion_accuracy: 0.5553 - emoti

Epoch 4: LearningRateScheduler setting learning rate to 1.805e-05.
Epoch 4/10
1837/1837 ━━━━━━━━━━━━━━━━━━━━ 0s 199ms/step - emotion_accuracy: 0.6069 - emotion
Epoch 4: val_emotion_accuracy did not improve from 0.54283
1837/1837 ━━━━━━━━━━━━━━━━━━━━ 387s 211ms/step - emotion_accuracy: 0.6069 - emoti

Epoch 5: LearningRateScheduler setting learning rate to 1.7147499999999998e-05.
Epoch 5/10
1837/1837 ━━━━━━━━━━━━━━━━━━━━ 0s 199ms/step - emotion_accuracy: 0.6379 - emotion
Epoch 5: val_emotion_accuracy improved from 0.54283 to 0.56357, saving model to b
1837/1837 ━━━━━━━━━━━━━━━━━━━━ 388s 211ms/step - emotion_accuracy: 0.6379 - emoti

Epoch 6: LearningRateScheduler setting learning rate to 1.6290125e-05.
Epoch 6/10
1837/1837 ━━━━━━━━━━━━━━━━━━━━ 0s 199ms/step - emotion_accuracy: 0.6537 - emotion
Epoch 6: val_emotion_accuracy improved from 0.56357 to 0.57529, saving model to b
1837/1837 ━━━━━━━━━━━━━━━━━━━━ 388s 211ms/step - emotion_accuracy: 0.6537 - emoti

Epoch 7: LearningRateScheduler setting learning rate to 1.5475618749999998e-05.
Epoch 7/10
1837/1837 ━━━━━━━━━━━━━━━━━━━━ 0s 199ms/step - emotion_accuracy: 0.6758 - emotion
Epoch 7: val_emotion_accuracy did not improve from 0.57529
1837/1837 ━━━━━━━━━━━━━━━━━━━━ 387s 211ms/step - emotion_accuracy: 0.6758 - emoti

Epoch 8: LearningRateScheduler setting learning rate to 1.4701837812499997e-05.
Epoch 8/10
1837/1837 ━━━━━━━━━━━━━━━━━━━━ 0s 199ms/step - emotion_accuracy: 0.6815 - emotion
```