Emotion Analysis and Detection in Mental Health Using Natural Language Processing Techniques

In [8]:
```python
import warnings
warnings.filterwarnings("ignore")
# Basic Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud

# Text Preprocessing
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# PyTorch Libraries
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from torch.optim import Adam

# Machine Learning
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder

# Gradio for Testing
import gradio as gr
```

2. Load and Explore the Dataset

In [17]:
```python
# Load the dataset
# Load the dataset
data = pd.read_csv('tweet_emotions.csv')

# Display the first few rows of the dataset
print("\nFirst 5 Rows of the Dataset:")
data.head()
```

```
First 5 Rows of the Dataset:
```

Out[17]:

	tweet_id	Emotion	Text
0	1956967341	empty	@tiffanylue i know i was listenin to bad habi...
1	1956967666	sadness	Layin n bed with a headache ughhhh...waitin o...
2	1956967696	sadness	Funeral ceremony...gloomy friday...
3	1956967789	enthusiasm	wants to hang out with friends SOON!
4	1956968416	neutral	@dannycastillo We want to trade with someone w...

In [18]:
```python
# View dataset information
print("Dataset Info:")
print(data.info())
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40000 entries, 0 to 39999
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   tweet_id  40000 non-null  int64 
 1   Emotion   40000 non-null  object
 2   Text      40000 non-null  object
dtypes: int64(1), object(2)
memory usage: 937.6+ KB
None
```

In [19]:
```python
# Display the shape of the dataset
print("\nDataset Shape:", data.shape)
```

```
Dataset Shape: (40000, 3)
```

In [ ]:

In [ ]:

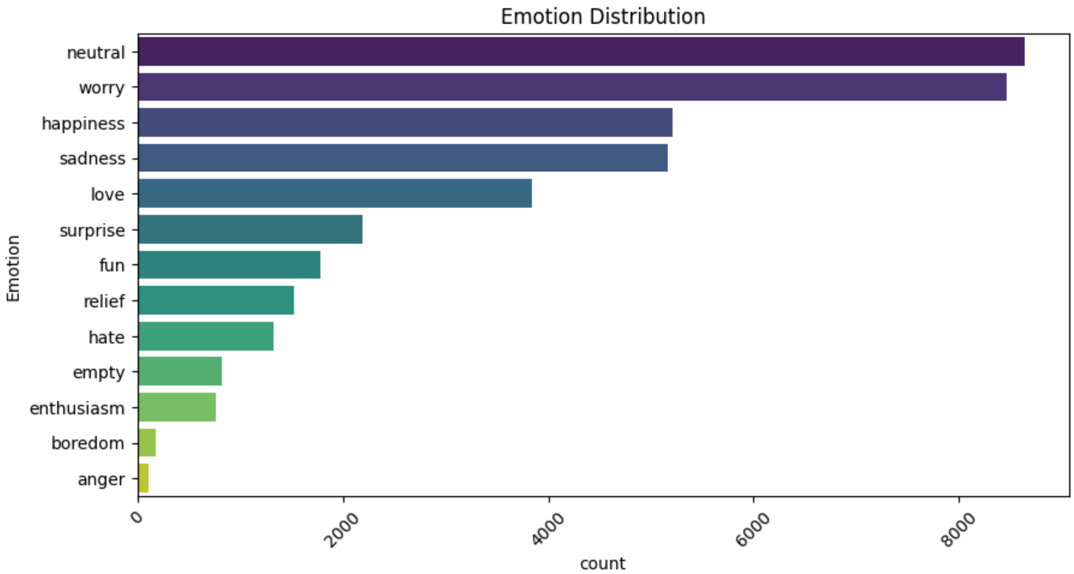
3. Text Preprocessing

In [20]:
```python
nltk.download('stopwords')
nltk.download('wordnet')

# Define text preprocessing steps
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def preprocess_text(text):
    text = text.lower()  # Convert to lowercase
    text = re.sub(r"http\S+|@\S+|#\S+", "", text)  # Remove URLs, mentions, hashtags
    text = re.sub(r"[^a-z\s]", "", text)  # Remove punctuation and numbers
```

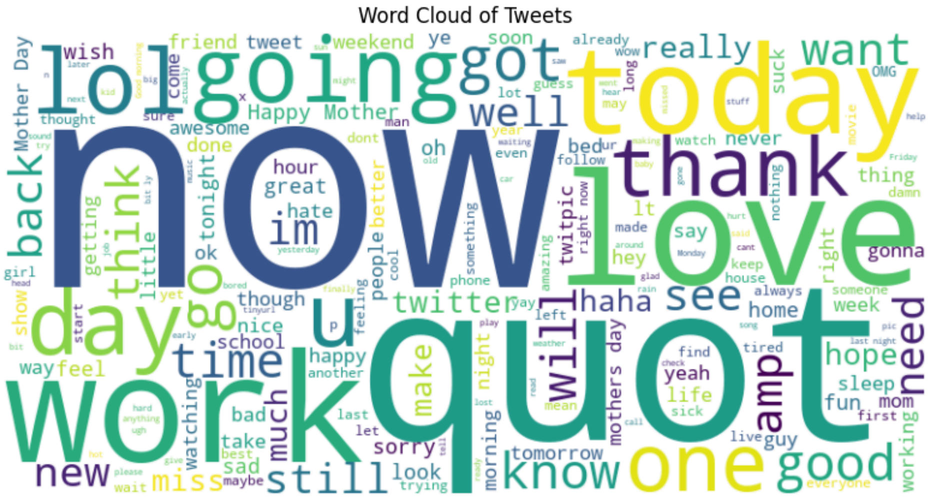
```python
    words = text.split()  # Tokenize
    words = [lemmatizer.lemmatize(word) for word in words if word not in stop_words]
    return " ".join(words)

# Apply preprocessing to the dataset
data['Cleaned_Text'] = data['Text'].apply(preprocess_text)

# Display sample processed text
print(data[['Text', 'Cleaned_Text']].head())
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\MAPILI\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\MAPILI\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
                                                Text  \
0  @tiffanylue i know  i was listenin to bad habi...
1  Layin n bed with a headache  ughhhh...waitin o...
2                Funeral ceremony...gloomy friday...
3               wants to hang out with friends SOON!
4  @dannycastillo We want to trade with someone w...

                                        Cleaned_Text
0  know listenin bad habit earlier started freaki...
1            layin n bed headache ughhhhwaitin call
2                     funeral ceremonygloomy friday
3                             want hang friend soon
4              want trade someone houston ticket one
```

In [21]:
```python
# Class distribution
plt.figure(figsize=(10, 5))
sns.countplot(data['Emotion'], order=data['Emotion'].value_counts().index, palette='viridis')
plt.title("Emotion Distribution")
plt.xticks(rotation=45)
plt.show()
```

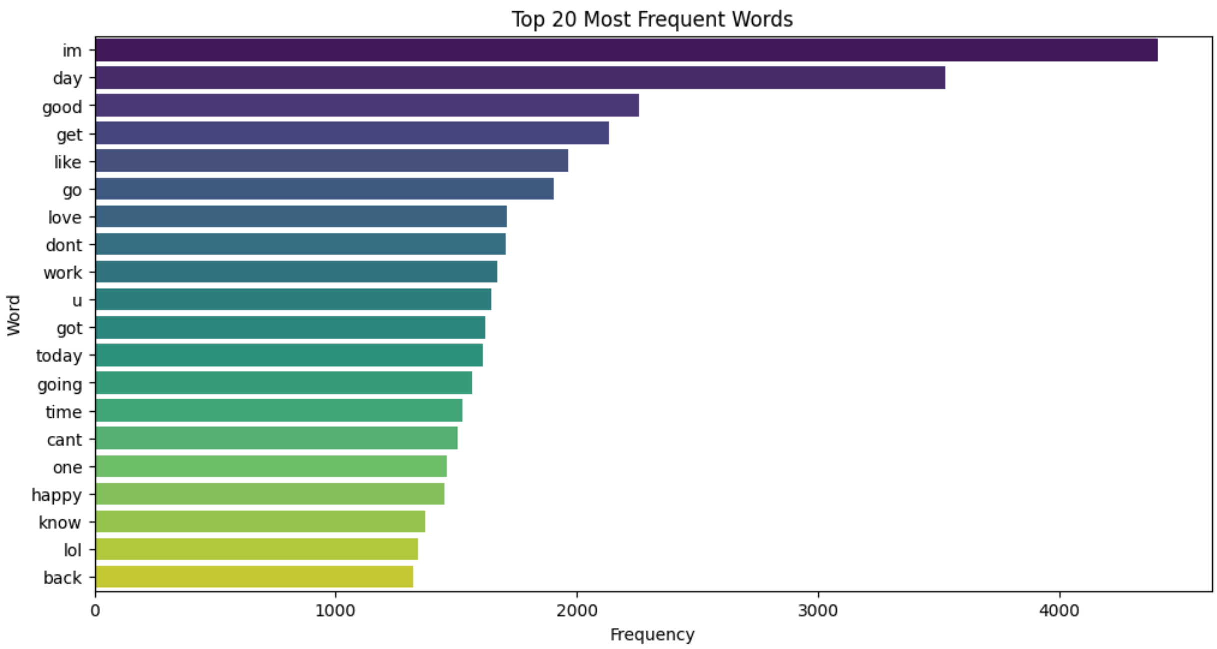


In [22]:
```python
# Generate a word cloud of the text
text = " ".join(data['Text'])
wordcloud = WordCloud(width=800, height=400, background_color='white').generate(text)

plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis('off')
plt.title("Word Cloud of Tweets")
plt.show()
```

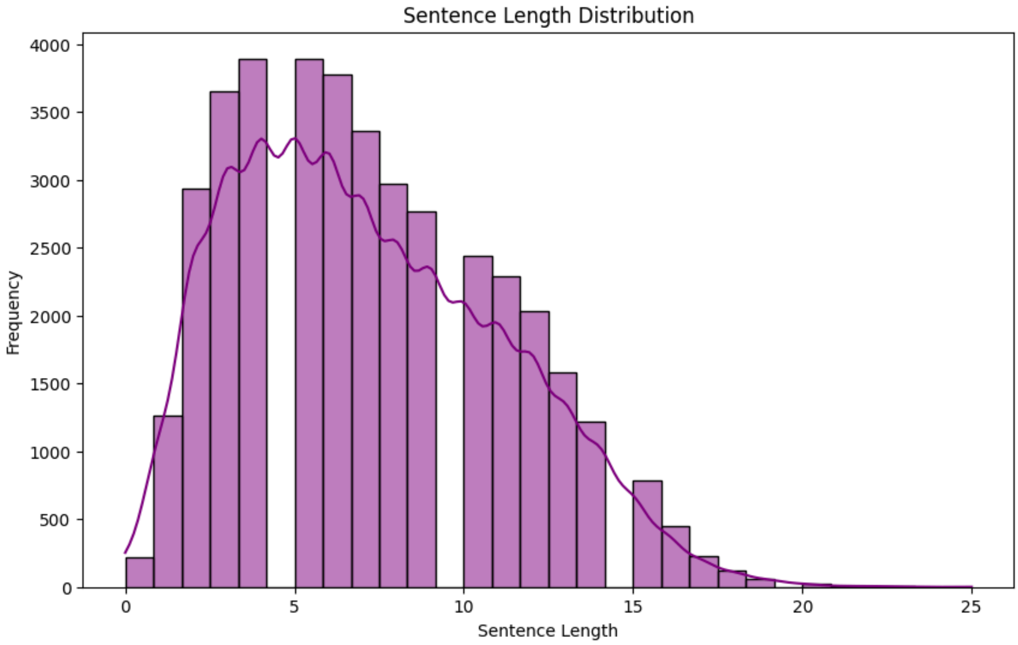


In [23]:
```python
from collections import Counter
from nltk.tokenize import word_tokenize
nltk.download('punkt')

# Combine all text
all_words = " ".join(data['Cleaned_Text']).split()

# Calculate word frequencies
word_freq = Counter(all_words)

# Convert to DataFrame
word_freq_df = pd.DataFrame(word_freq.most_common(20), columns=['Word', 'Frequency'])

# Plot the word frequencies
plt.figure(figsize=(12, 6))
sns.barplot(x='Frequency', y='Word', data=word_freq_df, palette='viridis')
plt.title("Top 20 Most Frequent Words")
plt.show()
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\MAPILI\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```



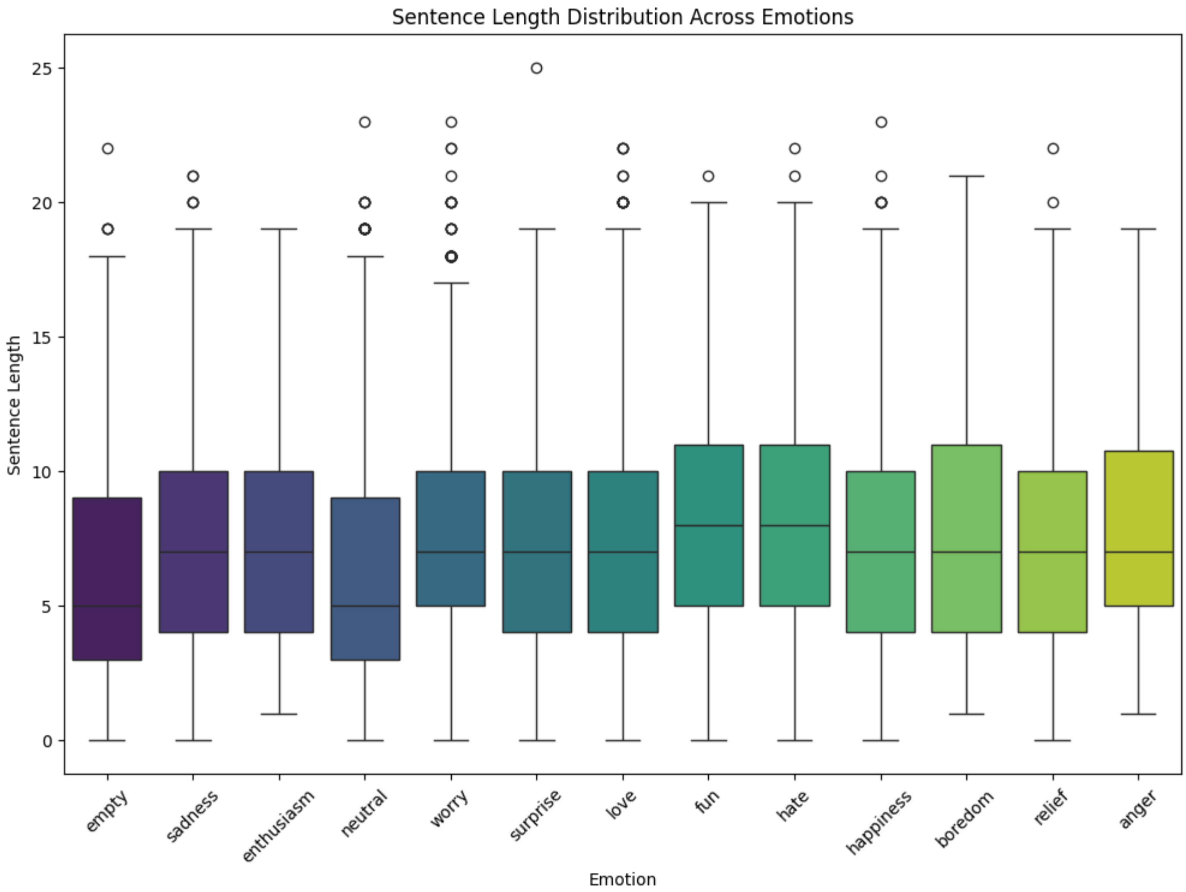
In [24]:
```python
# Calculate sentence lengths
data['Sentence_Length'] = data['Cleaned_Text'].apply(lambda x: len(word_tokenize(x)))

# Plot sentence length distribution
plt.figure(figsize=(10, 6))
sns.histplot(data['Sentence_Length'], bins=30, kde=True, color='purple')
plt.title("Sentence Length Distribution")
plt.xlabel("Sentence Length")
plt.ylabel("Frequency")
plt.show()
```



In [25]:
```
# Combine words by emotion
emotion_words = data.groupby('Emotion')['Cleaned_Text'].apply(lambda x: " ".join(x)).reset_index()

# Determine grid size based on the number of unique emotions
num_emotions = emotion_words.shape[0]
cols = 4  # Fixed number of columns
rows = (num_emotions // cols) + (num_emotions % cols > 0)  # Calculate required rows

# Generate word clouds for each emotion
plt.figure(figsize=(cols * 4, rows * 3))
for i, row in emotion_words.iterrows():
    plt.subplot(rows, cols, i + 1)  # Dynamically adjust rows and columns
    wordcloud = WordCloud(width=400, height=200, background_color='white').generate(row['Cleaned_Text'])
    plt.imshow(wordcloud, interpolation='bilinear')
    plt.axis('off')
    plt.title(f"Emotion: {row['Emotion']}")
plt.tight_layout()
plt.show()
```

In [26]:
```
# Plot emotion-wise sentence length distributions
plt.figure(figsize=(12, 8))
sns.boxplot(x='Emotion', y='Sentence_Length', data=data, palette='viridis')
plt.title("Sentence Length Distribution Across Emotions")
```

```python
plt.xticks(rotation=45)
plt.ylabel("Sentence Length")
plt.xlabel("Emotion")
plt.show()
```



In [27]:

```python
# Extract most frequent words for each emotion
emotion_word_freq = {}

for emotion in data['Emotion'].unique():
    words = " ".join(data[data['Emotion'] == emotion]['Cleaned_Text']).split()
    emotion_word_freq[emotion] = Counter(words).most_common(10)

# Display top words for each emotion
for emotion, freq in emotion_word_freq.items():
    print(f"Emotion: {emotion}")
    print("Top Words:")
    print(pd.DataFrame(freq, columns=['Word', 'Frequency']))
    print("\n")
```

```
Emotion: empty
Top Words:
    Word  Frequency
0     im         86
1   dont         47
2    get         44
3    day         44
4     go         40
5  bored         38
6   work         34
7   like         28
8   home         27
9    got         27


Emotion: sadness
Top Words:
    Word  Frequency
0     im        797
1    day        432
2    sad        373
3   miss        352
4   work        339
5     go        323
6    get        322
7   like        284
8  today        279
9   cant        275


Emotion: enthusiasm
Top Words:
    Word  Frequency
0     im         86
1     go         57
2   good         55
3   want         52
4    day         50
5    get         49
6      u         41
7   work         40
8   like         36
9  today         35


Emotion: neutral
Top Words:
    Word  Frequency
0     im        667
1    day        439
2    get        391
3     go        379
4   good        354
5   work        351
6   like        343
7    one        324
8  going        314
9      u        314


Emotion: worry
Top Words:
   Word  Frequency
0    im       1159
1   get        568
2  dont        546
3   day        541
4    go        477
5  cant        475
6  like        458
7  work        399
8   got        390
9  good        384


Emotion: surprise
Top Words:
   Word  Frequency
0    im        206
1   day        147
2   get        128
3    oh        116
4   got         99
5  like         99
6  cant         98
7  know         96
8  good         93
9     u         91


Emotion: love
Top Words:
     Word  Frequency
0    love        907
1     day        805
2  mother        600
3   happy        592
4      im        327
5    good        317
6     mom        220
```

```
7       u        213
8  thanks        189
9    like        187


Emotion: fun
Top Words:
   Word  Frequency
0    im        191
1   lol        177
2   fun        141
3   day        124
4  good        113
5  like        104
6  haha        102
7     u         99
8   get         96
9    go         93


Emotion: hate
Top Words:
     Word  Frequency
0    hate        231
1      im        152
2    suck         94
3    like         93
4    dont         88
5    work         86
6     get         86
7    cant         72
8     day         68
9  really         63


Emotion: happiness
Top Words:
     Word  Frequency
0     day        691
1    good        555
2      im        522
3   happy        406
4  thanks        296
5   great        292
6     lol        277
7   today        259
8     got        243
9    like        239


Emotion: boredom
Top Words:
     Word  Frequency
0      im         41
1   bored         27
2    work         15
3  really         12
4      go         12
5    like         12
6    dont         11
7   tired         10
8   still         10
9     amp         10


Emotion: relief
Top Words:
      Word  Frequency
0      day        172
1       im        165
2     good        124
3   thanks         91
4     time         85
5  finally         84
6    today         79
7     back         78
8     like         78
9      got         77


Emotion: anger
Top Words:
    Word  Frequency
0     im         12
1    get          9
2   dont          7
3    day          7
4   know          7
5     go          6
6   like          5
7    got          5
8   good          5
9  going          5
```

4. Data Preparation

In [28]:

```
# Encode labels
label_encoder = LabelEncoder()
data['Emotion'] = label_encoder.fit_transform(data['Emotion'])

# Train-test-validation split (80% train, 10% validation, 10% test)
X = data['Cleaned_Text']
y = data['Emotion']
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42, stratify=y_temp)

# Convert text to numerical representation using CountVectorizer
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer(max_features=10000)
X_train_vec = vectorizer.fit_transform(X_train).toarray()
X_val_vec = vectorizer.transform(X_val).toarray()
X_test_vec = vectorizer.transform(X_test).toarray()

# Convert to PyTorch tensors
X_train_tensor = torch.tensor(X_train_vec, dtype=torch.float32)
X_val_tensor = torch.tensor(X_val_vec, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test_vec, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train.values, dtype=torch.long)
y_val_tensor = torch.tensor(y_val.values, dtype=torch.long)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.long)

# PyTorch Dataset and DataLoader
class EmotionDataset(Dataset):
    def __init__(self, X, y):
        self.X = X
        self.y = y

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]

# Create datasets and data loaders for training, validation, and test
train_dataset = EmotionDataset(X_train_tensor, y_train_tensor)
val_dataset = EmotionDataset(X_val_tensor, y_val_tensor)
test_dataset = EmotionDataset(X_test_tensor, y_test_tensor)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

5. Define the PyTorch Model

In [29]:

```
class EmotionClassifier(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(EmotionClassifier, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.3)
        self.fc2 = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        x = self.fc1(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.fc2(x)
        return x

# Model Parameters
input_dim = X_train_vec.shape[1]
hidden_dim = 128
output_dim = len(label_encoder.classes_)

# Initialize Model, Loss, and Optimizer
model = EmotionClassifier(input_dim, hidden_dim, output_dim)
criterion = nn.CrossEntropyLoss()
optimizer = Adam(model.parameters(), lr=0.001)

# Move model to device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
```

Out[29]:

```
EmotionClassifier(
  (fc1): Linear(in_features=10000, out_features=128, bias=True)
  (relu): ReLU()
  (dropout): Dropout(p=0.3, inplace=False)
  (fc2): Linear(in_features=128, out_features=13, bias=True)
)
```

6. Train the Model

In [30]:

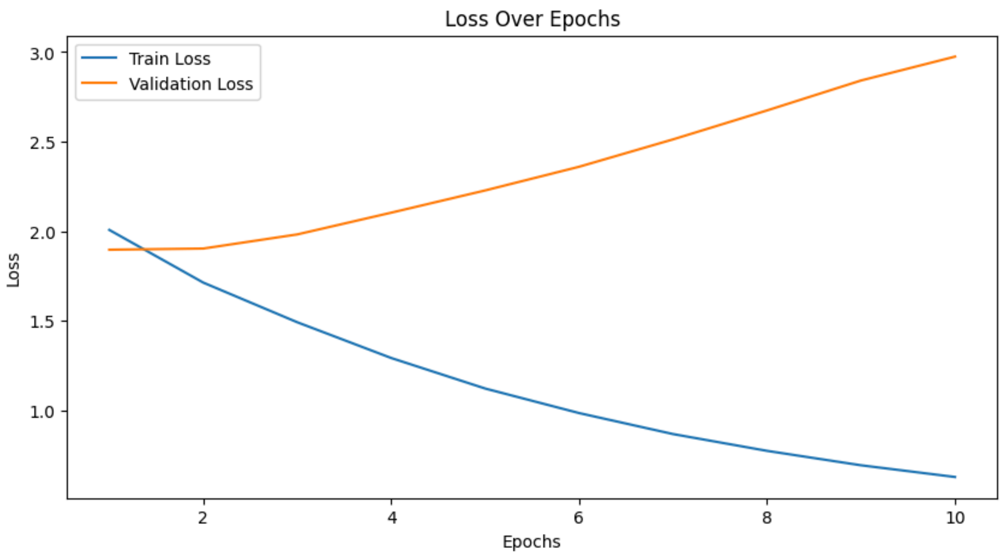
```
import matplotlib.pyplot as plt

# Initialize variables for tracking loss and accuracy
train_losses, val_losses = [], []
train_accuracies, val_accuracies = [], []  # Corrected initialization

# Training Loop
epochs = 10
for epoch in range(epochs):
    model.train()
    running_loss = 0.0
```

```python
    correct_train = 0
    total_train = 0

    # Training
    for X_batch, y_batch in train_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)

        optimizer.zero_grad()
        outputs = model(X_batch)
        loss = criterion(outputs, y_batch)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        correct_train += (predicted == y_batch).sum().item()
        total_train += y_batch.size(0)

    train_loss = running_loss / len(train_loader)
    train_accuracy = correct_train / total_train
    train_losses.append(train_loss)
    train_accuracies.append(train_accuracy)

    # Validation
    model.eval()
    running_val_loss = 0.0
    correct_val = 0
    total_val = 0

    with torch.no_grad():
        for X_val, y_val in val_loader:
            X_val, y_val = X_val.to(device), y_val.to(device)
            outputs = model(X_val)
            val_loss = criterion(outputs, y_val)
            running_val_loss += val_loss.item()

            _, predicted = torch.max(outputs, 1)
            correct_val += (predicted == y_val).sum().item()
            total_val += y_val.size(0)

    val_loss = running_val_loss / len(val_loader)
    val_accuracy = correct_val / total_val
    val_losses.append(val_loss)
    val_accuracies.append(val_accuracy)

    # Print epoch results
    print(f"Epoch {epoch+1}/{epochs}, Train Loss: {train_loss:.4f}, Train Accuracy: {train_accuracy:.4f}, "
          f"Val Loss: {val_loss:.4f}, Val Accuracy: {val_accuracy:.4f}")
```

```
Epoch 1/10, Train Loss: 2.0078, Train Accuracy: 0.3209, Val Loss: 1.8974, Val Accuracy: 0.3525
Epoch 2/10, Train Loss: 1.7138, Train Accuracy: 0.4228, Val Loss: 1.9040, Val Accuracy: 0.3538
Epoch 3/10, Train Loss: 1.4926, Train Accuracy: 0.5002, Val Loss: 1.9827, Val Accuracy: 0.3390
Epoch 4/10, Train Loss: 1.2931, Train Accuracy: 0.5686, Val Loss: 2.1041, Val Accuracy: 0.3285
Epoch 5/10, Train Loss: 1.1228, Train Accuracy: 0.6267, Val Loss: 2.2282, Val Accuracy: 0.3210
Epoch 6/10, Train Loss: 0.9858, Train Accuracy: 0.6748, Val Loss: 2.3604, Val Accuracy: 0.3185
Epoch 7/10, Train Loss: 0.8686, Train Accuracy: 0.7176, Val Loss: 2.5132, Val Accuracy: 0.3170
Epoch 8/10, Train Loss: 0.7750, Train Accuracy: 0.7499, Val Loss: 2.6747, Val Accuracy: 0.3123
Epoch 9/10, Train Loss: 0.6944, Train Accuracy: 0.7756, Val Loss: 2.8417, Val Accuracy: 0.3080
Epoch 10/10, Train Loss: 0.6292, Train Accuracy: 0.7973, Val Loss: 2.9749, Val Accuracy: 0.3090
```

In [31]:

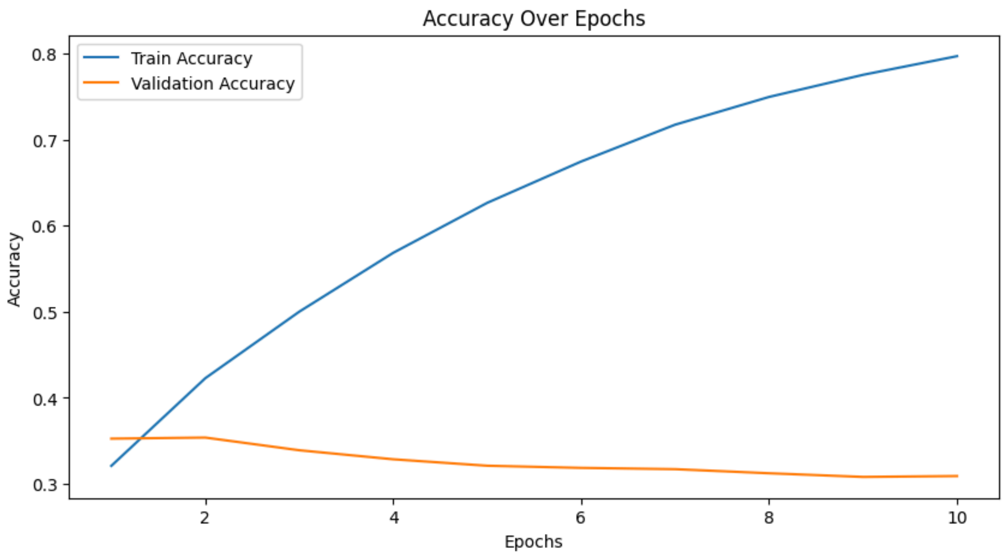
```python
# Plotting Training and Validation Loss
plt.figure(figsize=(10, 5))
plt.plot(range(1, epochs+1), train_losses, label='Train Loss')
plt.plot(range(1, epochs+1), val_losses, label='Validation Loss')
plt.title("Loss Over Epochs")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



In [32]:

```python
# Plotting Training and Validation Accuracy
plt.figure(figsize=(10, 5))
```

```python
plt.plot(range(1, epochs+1), train_accuracies, label='Train Accuracy')
plt.plot(range(1, epochs+1), val_accuracies, label='Validation Accuracy')
plt.title("Accuracy Over Epochs")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```

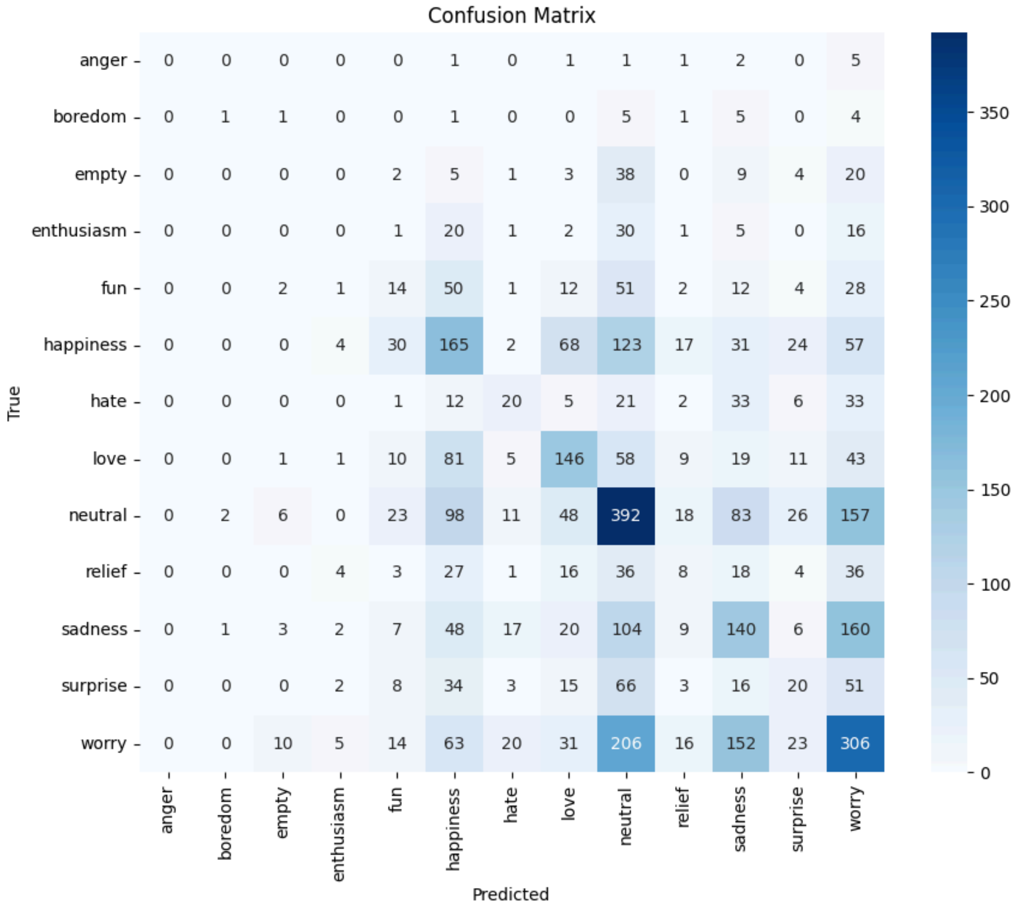


7. Evaluate the Model

In [33]:
```python
# Get the class labels as strings from the label encoder
class_labels = label_encoder.classes_

# Evaluation
model.eval()
y_pred = []
y_true = []

with torch.no_grad():
    for X_batch, y_batch in test_loader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)
        outputs = model(X_batch)
        _, preds = torch.max(outputs, 1)
        y_pred.extend(preds.cpu().numpy())
        y_true.extend(y_batch.cpu().numpy())

# Classification report
print("Classification Report:")
print(classification_report(y_true, y_pred, target_names=class_labels.astype(str)))
```

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

       anger       0.00      0.00      0.00        11
     boredom       0.25      0.06      0.09        18
       empty       0.00      0.00      0.00        82
  enthusiasm       0.00      0.00      0.00        76
         fun       0.12      0.08      0.10       177
   happiness       0.27      0.32      0.29       521
        hate       0.24      0.15      0.19       133
        love       0.40      0.38      0.39       384
     neutral       0.35      0.45      0.39       864
      relief       0.09      0.05      0.07       153
     sadness       0.27      0.27      0.27       517
    surprise       0.16      0.09      0.12       218
       worry       0.33      0.36      0.35       846

    accuracy                           0.30      4000
   macro avg       0.19      0.17      0.17      4000
weighted avg       0.28      0.30      0.29      4000
```

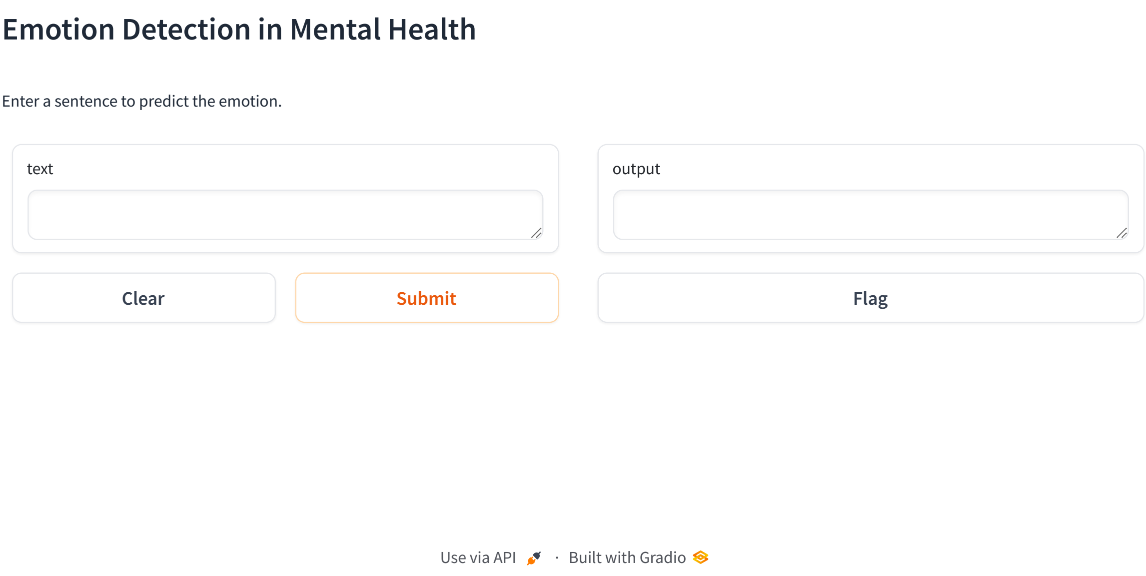
In [34]:
```python
# Confusion Matrix
conf_matrix = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```



8. Test the Model with Gradio

In [22]:
```python
# Gradio Interface
def predict_emotion(text):
    model.eval()
    processed_text = preprocess_text(text)
    vectorized_text = vectorizer.transform([processed_text]).toarray()
    input_tensor = torch.tensor(vectorized_text, dtype=torch.float32).to(device)
    with torch.no_grad():
        output = model(input_tensor)
        _, predicted = torch.max(output, 1)
    emotion = label_encoder.inverse_transform([predicted.cpu().numpy()[0]])
    return emotion[0]

iface = gr.Interface(
    fn=predict_emotion,
    inputs="text",
    outputs="text",
    title="Emotion Detection in Mental Health",
    description="Enter a sentence to predict the emotion."
)

iface.launch()
```

```
Running on local URL:  http://127.0.0.1:7863

To create a public link, set `share=True` in `launch()`.
```



Out[22]:

```
IMPORTANT: You are using gradio version 4.26.0, however version 4.44.1 is available, please upgrade.
--------
```

In [ ]:

Faster Emotion Prediction with DistilBERT for Real-time Applications

In [20]:

```python
import torch
from transformers import DistilBertTokenizer, DistilBertForSequenceClassification
import gradio as gr

# The pretrained DistilBERT model and tokenizer
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
model = DistilBertForSequenceClassification.from_pretrained('distilbert-base-uncased', num_labels=13)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

emotion_labels = [
    'happy', 'angry', 'sad', 'surprised', 'fearful', 'disgusted',
    'neutral', 'worry', 'happiness', 'sadness', 'love', 'surprise',
    'fun', 'relief', 'hate', 'empty', 'enthusiasm', 'boredom', 'anger'
]

# Define the prediction function
def predict_emotion(text):
    # Tokenize input text
    inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True, max_length=512)
    inputs = {key: value.to(device) for key, value in inputs.items()}

    # Get model predictions
    with torch.no_grad():
        outputs = model(**inputs)
        logits = outputs.logits
        predicted_class = torch.argmax(logits, dim=1).item()

    # Return the predicted emotion label
    return emotion_labels[predicted_class]

# Create the Gradio interface
iface = gr.Interface(
    fn=predict_emotion,  # Prediction function
    inputs=gr.Textbox(lines=2, placeholder="Enter your text here..."),  # User input (textbox)
    outputs=gr.Textbox(),  # Output predicted emotion
    title="Emotion Prediction in Mental Health",  # Title of the app
    description="Enter a sentence, and the model will predict the emotion.",  # Description
)

# Launch the Gradio interface
iface.launch()
```

```
Some weights of DistilBertForSequenceClassification were not initialized from the model checkpoint at distilbert-base-uncased and are newly initia
lized: ['classifier.bias', 'classifier.weight', 'pre_classifier.bias', 'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Running on local URL:  http://127.0.0.1:7861
IMPORTANT: You are using gradio version 4.26.0, however version 4.44.1 is available, please upgrade.
--------

To create a public link, set `share=True` in `launch()`.
```

Emotion Prediction in Mental Health

Enter a sentence, and the model will predict the emotion.

text

Enter your text here...

output

Flag

Clear

Submit

Use via API · Built with Gradio

Out[20]:

Pretrained Model (BERT) for Emotion Detection on Mental Health

In [22]:

```
import torch
from torch import nn
from torch.utils.data import Dataset, DataLoader
from transformers import BertTokenizer, BertForSequenceClassification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import re

# Ensure NLTK resources are downloaded
nltk.download('stopwords')
nltk.download('wordnet')

# Check if CUDA is available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

In [18]:

```
# Load Pretrained BERT Tokenizer and Model
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=13)  # Dynamically adjust number of labels later
model.to(device)
```

```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['cla
ssifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

Out[18]:
```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=13, bias=True)
)
```

In [10]:
```python
# Load the dataset
data = pd.read_csv('tweet_emotions.csv')

# Use the first 200 rows
data = data.head(200)

# Encode target labels (Emotion)
label_encoder = LabelEncoder()
data['Emotion'] = label_encoder.fit_transform(data['Emotion'])

# Preprocess the text data
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

def preprocess_text(text):
    text = re.sub(r"http\S+|www\S+|https\S+", '', text, flags=re.MULTILINE)  # Remove URLs
    text = re.sub(r'\@w+|\#', '', text)  # Remove @mentions and hashtags
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuations
    text = text.lower()  # Lowercase
    text = [lemmatizer.lemmatize(word) for word in text.split() if word not in stop_words]
    return ' '.join(text)

data['Text'] = data['Text'].apply(preprocess_text)

# Split into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(data['Text'], data['Emotion'], test_size=0.2, random_state=42)

# Define a custom dataset class
class TweetDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len=128):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        text = self.texts.iloc[idx]
        label = self.labels.iloc[idx]
        encoding = self.tokenizer.encode_plus(
            text,
            add_special_tokens=True,
            max_length=self.max_len,
            return_token_type_ids=False,
            padding='max_length',
            truncation=True,
            return_attention_mask=True,
            return_tensors='pt',
        )
        return {
            'input_ids': encoding['input_ids'].squeeze(0),
            'attention_mask': encoding['attention_mask'].squeeze(0),
```

```
            'label': torch.tensor(label, dtype=torch.long)
        }

# Create Datasets and DataLoaders
train_dataset = TweetDataset(X_train, y_train, tokenizer)
val_dataset = TweetDataset(X_val, y_val, tokenizer)

train_loader = DataLoader(train_dataset, batch_size=8, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=8, shuffle=False)

# Define training parameters
optimizer = torch.optim.AdamW(model.parameters(), lr=2e-5)
criterion = nn.CrossEntropyLoss()
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\MAPILI\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\MAPILI\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['cla
ssifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

In [11]:

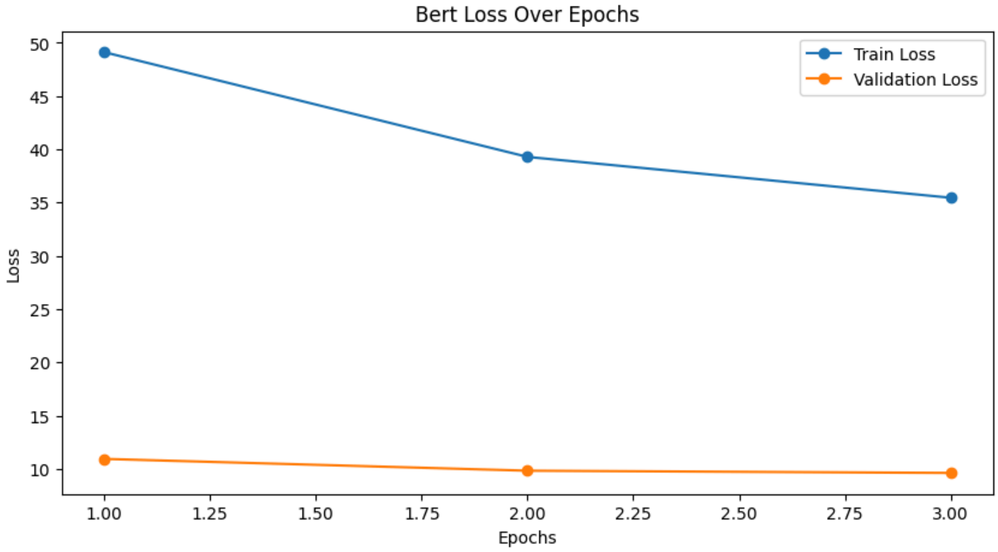
```
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, classification_report

# Initialize lists to store metrics
train_losses = []
val_losses = []
train_accuracies = []
val_accuracies = []

# Training loop with metrics tracking
def train_model(model, train_loader, val_loader, criterion, optimizer, device, epochs=3):
    for epoch in range(epochs):
        # Training phase
        model.train()
        total_train_loss = 0
        correct_train = 0
        for batch in train_loader:
            optimizer.zero_grad()
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['label'].to(device)

            outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labels)
            loss = outputs.loss
            logits = outputs.logits
            total_train_loss += loss.item()
            loss.backward()
            optimizer.step()

            preds = torch.argmax(logits, dim=1)
            correct_train += (preds == labels).sum().item()

        train_accuracy = correct_train / len(train_loader.dataset)
        train_losses.append(total_train_loss)
        train_accuracies.append(train_accuracy)
        print(f"Epoch {epoch+1}/{epochs} | Train Loss: {total_train_loss:.4f} | Train Accuracy: {train_accuracy:.4f}")

        # Validation phase
        model.eval()
        total_val_loss = 0
        correct_val = 0
        with torch.no_grad():
            for batch in val_loader:
                input_ids = batch['input_ids'].to(device)
                attention_mask = batch['attention_mask'].to(device)
                labels = batch['label'].to(device)

                outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labels)
                loss = outputs.loss
                logits = outputs.logits
                total_val_loss += loss.item()

                preds = torch.argmax(logits, dim=1)
                correct_val += (preds == labels).sum().item()

        val_accuracy = correct_val / len(val_loader.dataset)
        val_losses.append(total_val_loss)
        val_accuracies.append(val_accuracy)
        print(f"Validation Loss: {total_val_loss:.4f} | Validation Accuracy: {val_accuracy:.4f}")

# Train the model
epochs = 3
train_model(model, train_loader, val_loader, criterion, optimizer, device, epochs)
```

```
Epoch 1/3 | Train Loss: 49.1179 | Train Accuracy: 0.1875
Validation Loss: 10.9392 | Validation Accuracy: 0.1750
Epoch 2/3 | Train Loss: 39.2878 | Train Accuracy: 0.3063
Validation Loss: 9.8322 | Validation Accuracy: 0.1750
Epoch 3/3 | Train Loss: 35.4526 | Train Accuracy: 0.4437
Validation Loss: 9.6245 | Validation Accuracy: 0.3500
```

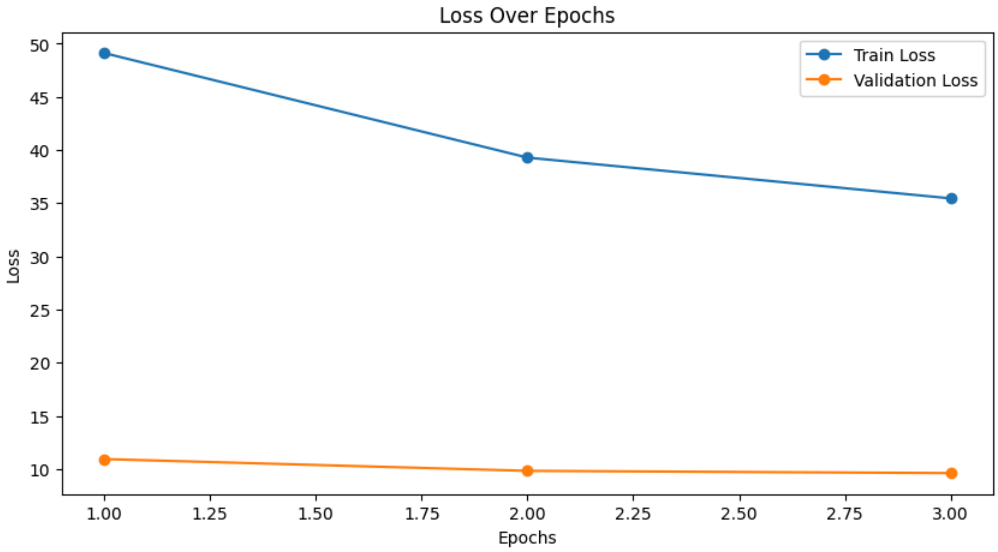
In [19]:

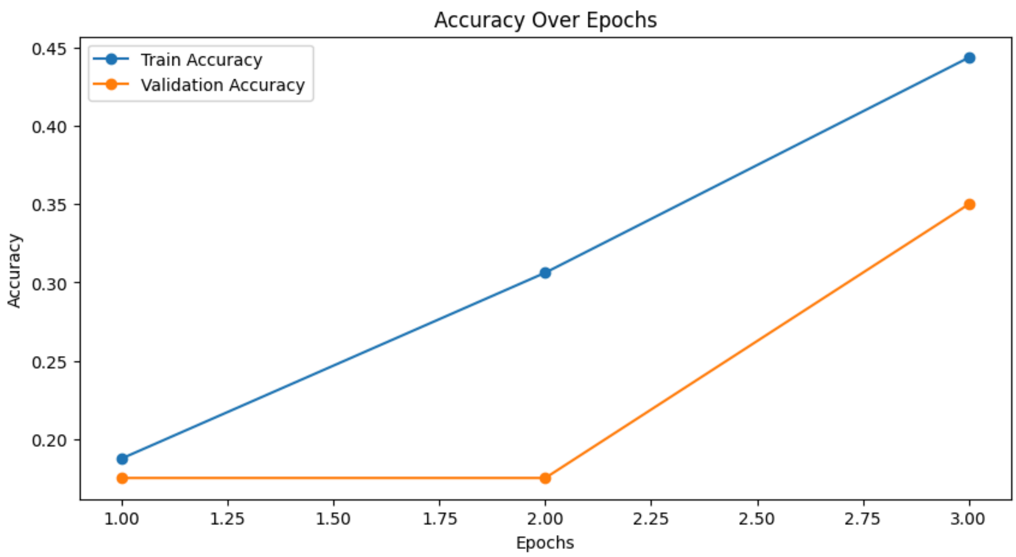
```
# Plotting Training and Validation Loss
plt.figure(figsize=(10, 5))
plt.plot(range(1, epochs+1), train_losses, label='Train Loss', marker='o')
plt.plot(range(1, epochs+1), val_losses, label='Validation Loss', marker='o')
plt.title("Bert Loss Over Epochs")
plt.xlabel("Epochs")
```

```python
plt.ylabel("Loss")
plt.legend()
plt.show()
```



In [12]:
```python
# Plotting Training and Validation Loss
plt.figure(figsize=(10, 5))
plt.plot(range(1, epochs+1), train_losses, label='Train Loss', marker='o')
plt.plot(range(1, epochs+1), val_losses, label='Validation Loss', marker='o')
plt.title("Bert Loss Over Epochs")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()

# Plotting Training and Validation Accuracy
plt.figure(figsize=(10, 5))
plt.plot(range(1, epochs+1), train_accuracies, label='Train Accuracy', marker='o')
plt.plot(range(1, epochs+1), val_accuracies, label='Validation Accuracy', marker='o')
plt.title("Accuracy Over Epochs")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
plt.show()
```





In [15]:
```
# Test set evaluation
test_loader = val_loader  # Using validation data as test data for simplicity in this example
model.eval()
all_preds = []
all_labels = []

with torch.no_grad():
    for batch in test_loader:
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['label'].to(device)

        # Forward pass
        outputs = model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits

        _, predicted = torch.max(logits, 1)
        all_preds.extend(predicted.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# Calculate accuracy
test_accuracy = accuracy_score(all_labels, all_preds)
print(f"Test Accuracy: {test_accuracy:.4f}")
```

```
Test Accuracy: 0.3500
```

In [16]:
```
num_classes = 13  # Set to 13 for the 13 classes in your dataset
labels = list(range(num_classes))  # This creates a list of integers from 0 to 12 for a 13-class problem

# Classification report
report = classification_report(all_labels, all_preds, labels=labels, target_names=[
    'Empty', 'Sadness', 'Enthusiasm', 'Neutral', 'Worry', 'Surprise',
    'Love', 'Fun', 'Hate', 'Happiness', 'Boredom', 'Relief', 'Anger'
])
print("Classification Report:")
print(report)
```

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

       Empty       0.00      0.00      0.00         0
     Sadness       0.00      0.00      0.00         0
  Enthusiasm       0.00      0.00      0.00         1
     Neutral       0.00      0.00      0.00         3
       Worry       0.00      0.00      0.00         1
    Surprise       0.00      0.00      0.00         2
        Love       0.00      0.00      0.00         1
         Fun       0.33      0.86      0.48         7
        Hate       0.00      0.00      0.00         0
   Happiness       0.25      0.17      0.20        12
     Boredom       0.00      0.00      0.00         1
      Relief       0.43      0.50      0.46        12
       Anger       0.00      0.00      0.00         0

   micro avg       0.35      0.35      0.35        40
   macro avg       0.08      0.12      0.09        40
weighted avg       0.26      0.35      0.28        40
```

In [ ]:

In [17]:
```
import gradio as gr
import torch
from transformers import BertTokenizer, BertForSequenceClassification

# Load pretrained BERT model and tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=6)  # Assuming 6 emotion classes
model.to(device)

# Emotion labels (replace with actual labels you are using in your project)
emotion_labels = ['happy', 'angry', 'sad', 'surprised', 'fearful', 'disgusted']
```

```python
# Define the prediction function
def predict_emotion(text):
    # Tokenize input text
    inputs = tokenizer(text, return_tensors="pt", padding=True, truncation=True, max_length=512)
    inputs = {key: value.to(device) for key, value in inputs.items()}

    # Get model predictions
    with torch.no_grad():
        outputs = model(**inputs)
        logits = outputs.logits
        predicted_class = torch.argmax(logits, dim=1).item()

    # Return the predicted emotion label
    return emotion_labels[predicted_class]

# Create the Gradio interface
iface = gr.Interface(
    fn=predict_emotion,  # Prediction function
    inputs=gr.Textbox(lines=2, placeholder="Enter your text here..."),  # User input (textbox)
    outputs=gr.Textbox(),  # Output predicted emotion
    title="Emotion Prediction in Mental health",  # Title of the app
    description="Enter a sentence, and the model will predict the emotion.",  # Description
)

# Launch the Gradio interface
iface.launch()
```

```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['cla
ssifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Running on local URL:  http://127.0.0.1:7860

To create a public link, set `share=True` in `launch()`.
```

Emotion Prediction in Mental health

Enter a sentence, and the model will predict the emotion.

text

Enter your text here...

output

Clear

Submit

Flag

Use via API · Built with Gradio

Out[17]:

```
IMPORTANT: You are using gradio version 4.26.0, however version 4.44.1 is available, please upgrade.
--------
```

In [ ]:

In [ ]: