Module 7: Sentiment and Emotion Analysis Lab

ITAI 2373 - Natural Language Processing

Lab Overview

Welcome to the Sentiment and Emotion Analysis lab! In this hands-on session, you'll build a complete emotion detection system that works with both text and speech. This lab connects directly to Module 7's concepts and gives you practical experience with real-world emotion analysis.

What You'll Build Today:

1. **Text Sentiment Analyzer** using VADER and TextBlob
2. **Machine Learning Classifier** with scikit-learn
3. **Speech Emotion Detector** using audio features
4. **Multimodal System** combining text and speech analysis

Real Data You'll Use:

- Customer reviews from multiple domains
- Social media posts with emotion labels
- Audio recordings with emotional speech
- Multimodal datasets combining text and audio

Learning Objectives:

By the end of this lab, you will:

- Understand the differences between rule-based and ML approaches to sentiment analysis
- Build and evaluate multiple sentiment analysis systems
- Extract and analyze emotional features from speech
- Create a multimodal emotion detection system
- Critically evaluate bias and fairness in emotion detection systems

Before we start let's see how this lab ties to previous modules

---

🔗 Building on Previous Modules

Welcome to Module 7! This lab builds directly on everything you've learned so far:

From Module 1 (Introduction to NLP):

- We'll apply NLP to understand human emotions and opinions
- Real-world applications: customer feedback, social media monitoring, healthcare

From Module 2 (Text Preprocessing):

- We'll use tokenization, normalization, and cleaning techniques
- Preprocessing becomes crucial for emotion detection accuracy

From Module 3 (Audio and Preprocessing):

- We'll extract emotional features from speech signals
- Combine text and audio for multimodal emotion analysis

From Module 4 (Text Representation):

- We'll use TF-IDF vectorization for machine learning approaches
- Compare rule-based vs. ML feature representations

From Module 5 (Part-of-Speech Tagging):

- POS tags help identify emotional intensity (adjectives, adverbs)
- Grammatical patterns reveal sentiment structure

From Module 6 (Syntax and Parsing):

- Dependency relationships show emotional targets ("I love THIS product")
- Syntactic patterns help resolve sentiment scope and negation

🚀 Now, Let's Get Started!

Run each cell in order and complete the exercises marked with 🖍️ **YOUR TURN**

Part 0: Setup and Installation

First, let's install all the libraries we'll need for this comprehensive lab. We'll be working with text processing, machine learning, and audio analysis libraries.

```
# Install required libraries for Google Colab
!pip install vaderSentiment textblob librosa soundfile
!python -m textblob.download_corpora
```

```
Requirement already satisfied: vaderSentiment in /usr/local/lib/python3.11/dist-packages (3.3.2)
Requirement already satisfied: textblob in /usr/local/lib/python3.11/dist-packages (0.19.0)
Requirement already satisfied: librosa in /usr/local/lib/python3.11/dist-packages (0.11.0)
Requirement already satisfied: soundfile in /usr/local/lib/python3.11/dist-packages (0.13.1)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from vaderSentiment) (2.32.3)
Requirement already satisfied: nltk>=3.9 in /usr/local/lib/python3.11/dist-packages (from textblob) (3.9.1)
Requirement already satisfied: audioread>=2.1.9 in /usr/local/lib/python3.11/dist-packages (from librosa) (3.0.1)
Requirement already satisfied: numba>=0.51.0 in /usr/local/lib/python3.11/dist-packages (from librosa) (0.60.0)
Requirement already satisfied: numpy>=1.22.3 in /usr/local/lib/python3.11/dist-packages (from librosa) (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from librosa) (1.16.0)
Requirement already satisfied: scikit-learn>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from librosa) (1.6.1)
Requirement already satisfied: joblib>=1.0 in /usr/local/lib/python3.11/dist-packages (from librosa) (1.5.1)
Requirement already satisfied: decorator>=4.3.0 in /usr/local/lib/python3.11/dist-packages (from librosa) (4.4.2)
Requirement already satisfied: pooch>=1.1 in /usr/local/lib/python3.11/dist-packages (from librosa) (1.8.2)
Requirement already satisfied: soxr>=0.3.2 in /usr/local/lib/python3.11/dist-packages (from librosa) (0.5.0.post1)
Requirement already satisfied: typing_extensions>=4.1.1 in /usr/local/lib/python3.11/dist-packages (from librosa) (4.14.1)
Requirement already satisfied: lazy_loader>=0.1 in /usr/local/lib/python3.11/dist-packages (from librosa) (0.4)
Requirement already satisfied: msgpack>=1.0 in /usr/local/lib/python3.11/dist-packages (from librosa) (1.1.1)
Requirement already satisfied: cffi>=1.0 in /usr/local/lib/python3.11/dist-packages (from soundfile) (1.17.1)
Requirement already satisfied: pycparser in /usr/local/lib/python3.11/dist-packages (from cffi>=1.0->soundfile) (2.22)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from lazy_loader>=0.1->librosa) (25.0)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages (from nltk>=3.9->textblob) (8.2.1)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11/dist-packages (from nltk>=3.9->textblob) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from nltk>=3.9->textblob) (4.67.1)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.11/dist-packages (from numba>=0.51.0->librosa) (0.43
Requirement already satisfied: platformdirs>=2.5.0 in /usr/local/lib/python3.11/dist-packages (from pooch>=1.1->librosa) (4.3.8)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->vaderSentiment) (3.4.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->vaderSentiment) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->vaderSentiment) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->vaderSentiment) (2025.7.14)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn>=1.1.0->librosa) (3.6.
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Package brown is already up-to-date!
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger_eng is already up-to-
[nltk_data]       date!
[nltk_data] Downloading package conll2000 to /root/nltk_data...
[nltk_data]   Package conll2000 is already up-to-date!
[nltk_data] Downloading package movie_reviews to /root/nltk_data...
[nltk_data]   Package movie_reviews is already up-to-date!
Finished.
```

```
# Import all necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.preprocessing import StandardScaler
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
from textblob import TextBlob
```

```
import librosa
import soundfile as sf
from IPython.display import Audio, display
import warnings
warnings.filterwarnings('ignore')

# Set up plotting style
plt.style.use('default')
sns.set_palette("husl")

print("✅ All libraries installed and imported successfully!")
print("📚 Ready to start building emotion detection systems!")
```

```
✅ All libraries installed and imported successfully!
📚 Ready to start building emotion detection systems!
```

🔗 Let's Review in Practice How Previous Modules Tie in to Today:

```
# Example text that demonstrates concepts from all previous modules
example_text = "I absolutely LOVE this amazing product! It works perfectly and exceeded my expectations."

print("🔍 Analyzing with Previous Module Concepts:")
print(f"Original text: {example_text}")
print()

# Module 2: Text Preprocessing
import re
import string

def basic_preprocess(text):
    # Tokenization (Module 2)
    tokens = text.split()
    # Normalization (Module 2)
    tokens = [token.lower().strip(string.punctuation) for token in tokens]
    return tokens

tokens = basic_preprocess(example_text)
print(f"📝 Module 2 (Preprocessing) - Tokens: {tokens}")

# Module 4: Text Representation (TF-IDF preview)
sample_docs = [
    "I love this product",
    "This product is amazing",
    "I hate this terrible product"
]

vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform(sample_docs)
feature_names = vectorizer.get_feature_names_out()

print(f"\n🔢 Module 4 (Text Representation) - TF-IDF Features: {list(feature_names)}")

# Module 5: POS Tagging for sentiment
from textblob import TextBlob
blob = TextBlob(example_text)
pos_tags = blob.tags

print(f"\n🏷️ Module 5 (POS Tagging) - Tags: {pos_tags}")
print("   Notice: Adjectives (JJ) often carry emotional weight: 'amazing', 'perfect'")

# Module 6: Syntax for sentiment scope
print(f"\n🕵️ Module 6 (Syntax/Parsing) - Dependency relationships help us understand:")
print("   - What is being loved? (the product)")
print("   - What makes it amazing? (it works perfectly)")
print("   - Scope of sentiment: entire product experience")

print("\n💡 Key Insight: Sentiment analysis combines ALL previous concepts!")
print("   - Preprocessing cleans the text")
print("   - POS tags identify emotional words")
print("   - Syntax shows what the emotions target")
print("   - Text representation enables ML approaches")
```

```
🔍 Analyzing with Previous Module Concepts:
Original text: I absolutely LOVE this amazing product! It works perfectly and exceeded my expectations.
```

```
📝 Module 2 (Preprocessing) - Tokens: ['i', 'absolutely', 'love', 'this', 'amazing', 'product', 'it', 'works', 'perfectly', 'and', 'exc

🔢 Module 4 (Text Representation) - TF-IDF Features: ['amazing', 'hate', 'is', 'love', 'product', 'terrible', 'this']

🏷️ Module 5 (POS Tagging) - Tags: [('I', 'PRP'), ('absolutely', 'RB'), ('LOVE', 'VBP'), ('this', 'DT'), ('amazing', 'JJ'), ('product',
   Notice: Adjectives (JJ) often carry emotional weight: 'amazing', 'perfect'

🕵️ Module 6 (Syntax/Parsing) - Dependency relationships help us understand:
   - What is being loved? (the product)
   - What makes it amazing? (it works perfectly)
   - Scope of sentiment: entire product experience

💡 Key Insight: Sentiment analysis combines ALL previous concepts!
   - Preprocessing cleans the text
   - POS tags identify emotional words
   - Syntax shows what the emotions target
   - Text representation enables ML approaches
```

📚 Let's Recap some of the foundation from EDA & Machine Learning DataFrames Quick Review

What is a DataFrame?

Think of a **DataFrame** as a digital spreadsheet or table, just like Excel. It has:

- **Rows** (horizontal) - each row represents one piece of data
- **Columns** (vertical) - each column represents a different attribute or feature
- **Cells** - where specific data values are stored

In Python, we use **pandas** (imported as `pd`) to create and work with DataFrames.

Step-by-Step Breakdown of Our Sample Data Creation:

Step 1: Creating the Text Data

```
sample_texts = [
    "I absolutely love this product! It's amazing and works perfectly.",
    "This is the worst purchase I've ever made. Completely disappointed.",
    # ... more examples
]
```

What's happening here:

- We create a **list** (think of it as a container) of text examples
- Each text represents what a real customer might write in a review or social media post
- The square brackets `[]` define a list in Python
- Each text is in quotes because it's a **string** (text data type)

Why these specific texts:

- **Positive examples**: "I absolutely love this product!" (clearly happy/positive)
- **Negative examples**: "This is the worst purchase!" (clearly angry/negative)
- **Neutral examples**: "The product is okay" (neither positive nor negative)
- **Social media style**: "OMG this is INCREDIBLE!!! 😍" (with emojis and excitement)

Step 2: Creating the DataFrame Structure

```
df_sample = pd.DataFrame({
    'text': sample_texts,
    'source': ['review', 'review', 'social_media', ...]
})
```

What's happening here:

- `pd.DataFrame({...})` creates a new table/spreadsheet
- The curly braces `{}` define a **dictionary** - a way to pair names with values
- `'text': sample_texts` creates a column called "text" and fills it with our text examples
- `'source': [...]` creates a column called "source" and labels each text as either "review" or "social_media"

The resulting table looks like this:

Index	text	source
0	"I absolutely love this product!..."	review
1	"This is the worst purchase..."	review
2	"The product is okay..."	review
3	"OMG this is INCREDIBLE!!!"	social_media
4	"Meh... it's fine I guess..."	review

🎯 Key Concepts Revisited:

1. **Lists** `[]`: Store multiple items in order
2. **Dictionaries** `{}`: Pair names with values (like "name": "John")
3. **DataFrames**: Tables for organizing data with rows and columns
4. **Columns**: Vertical data categories (like "text" and "source")
5. **Rows**: Horizontal data entries (each customer comment)

🔍 Why Create This Sample Data?

1. Controlled Testing Environment

- We know exactly what emotions each text should express
- We can test if our emotion detection systems work correctly
- Like having an answer key to check our work

2. Diverse Text Types

- **Product reviews**: Professional, detailed feedback
- **Social media posts**: Casual, with emojis and slang
- This tests how well our systems handle different writing styles

3. Balanced Emotions

- Mix of positive, negative, and neutral sentiments
- Ensures our system can detect all types of emotions
- Prevents bias toward one emotion type

4. Real-World Simulation

- These texts represent what you'd actually find online
- Helps us understand how emotion detection works in practice
- Prepares us for analyzing real customer feedback

🚀 Why This Matters for Emotion Analysis:

This sample data serves as our **training ground** where we can:

- Test different emotion detection methods
- Compare their accuracy
- Understand their strengths and weaknesses
- Learn how to handle different types of text

Think of it like practicing basketball shots on a practice court before playing in a real game - we need controlled examples before analyzing real customer feedback!

📊 Understanding the Output Code:

```
print(f"Total texts: {len(df_sample)}")
print(f"Sources: {df_sample['source'].value_counts().to_dict()}")
```

What each line does:

- `len(df_sample)` counts how many rows (text examples) we have
- `df_sample['source'].value_counts()` counts how many of each source type we have
- `.to_dict()` converts the count into a dictionary format for easy reading

This gives us a quick summary of our data structure and helps us verify everything was created correctly

```
# Create sample data that demonstrates different linguistic phenomena
sample_texts = [
    "I absolutely love this product! It's amazing and works perfectly.",  # Positive with intensifiers
    "This is the worst purchase I've ever made. Completely disappointed.",  # Negative with superlatives
    "The product is okay, nothing special but does the job.",  # Neutral with mixed signals
    "OMG this is INCREDIBLE!!! 😍 Best thing ever!!!",  # Social media style
    "I don't hate this product, but it's not great either.",  # Negation (Module 6 syntax!)
    "The customer service was not bad, actually quite helpful.",  # Double negation
    "This expensive product should work better for the price.",  # Implicit criticism
    "Fast shipping, good quality, would recommend to others.",  # Multiple aspects
    "It's fine I guess... could be worse 😐",  # Lukewarm sentiment
    "TERRIBLE quality!!! Waste of money 😡 Never buying again!",  # Strong negative
]

# Create DataFrame
df_sample = pd.DataFrame({
    'text': sample_texts,
    'linguistic_feature': [
        'intensifiers', 'superlatives', 'mixed_signals', 'social_media',
        'negation', 'double_negation', 'implicit', 'multi_aspect',
        'lukewarm', 'strong_negative'
    ]
})

print("📝 Sample Data with Linguistic Challenges:")
for i, row in df_sample.iterrows():
    print(f"{i+1}. [{row['linguistic_feature']}] {row['text']}")

print("📝 Sample Data Created:")
print(f"Total texts: {len(df_sample)}")
print(f"Linguistic Features: {df_sample['linguistic_feature'].value_counts().to_dict()}")
print("\n🔍 First few examples:")
for i, row in df_sample.head(3).iterrows():
    print(f"{i+1}. [{row['linguistic_feature']}] {row['text']}")
```

```
📝 Sample Data with Linguistic Challenges:
1. [intensifiers] I absolutely love this product! It's amazing and works perfectly.
2. [superlatives] This is the worst purchase I've ever made. Completely disappointed.
3. [mixed_signals] The product is okay, nothing special but does the job.
4. [social_media] OMG this is INCREDIBLE!!! 😍 Best thing ever!!!
5. [negation] I don't hate this product, but it's not great either.
6. [double_negation] The customer service was not bad, actually quite helpful.
7. [implicit] This expensive product should work better for the price.
8. [multi_aspect] Fast shipping, good quality, would recommend to others.
9. [lukewarm] It's fine I guess... could be worse 😐
10. [strong_negative] TERRIBLE quality!!! Waste of money 😡 Never buying again!
📝 Sample Data Created:
Total texts: 10
Linguistic Features: {'intensifiers': 1, 'superlatives': 1, 'mixed_signals': 1, 'social_media': 1, 'negation': 1, 'double_negation': 1,

🔍 First few examples:
1. [intensifiers] I absolutely love this product! It's amazing and works perfectly.
2. [superlatives] This is the worst purchase I've ever made. Completely disappointed.
3. [mixed_signals] The product is okay, nothing special but does the job.
```

Part 1: Text Sentiment Analysis with VADER and TextBlob

Understanding Rule-Based Sentiment Analysis

Rule-based sentiment analysis uses **lexicons** (dictionaries) that map words to sentiment scores. This connects directly to Module 4's discussion of text representation - instead of TF-IDF vectors, we use pre-built emotional dictionaries.

VADER (Valence Aware Dictionary and sEntiment Reasoner):

- Designed for social media text
- Handles emojis, capitalization, punctuation
- Uses linguistic rules (like Module 6's parsing rules)

TextBlob:

- General-purpose sentiment analysis
- Provides both polarity and subjectivity scores
- Based on movie review corpus

1.1 VADER Sentiment Analysis

VADER (Valence Aware Dictionary and sEntiment Reasoner) is specifically designed for social media text. It:

- Handles emojis, slang, and intensifiers
- Provides compound scores from -1 (most negative) to +1 (most positive)
- Gives detailed breakdowns (positive, negative, neutral)

Let's see how VADER analyzes our sample texts:

```
# Initialize VADER analyzer
vader_analyzer = SentimentIntensityAnalyzer()

def analyze_with_vader(text):
    """Analyze sentiment using VADER"""
    scores = vader_analyzer.polarity_scores(text)
    return scores

def classify_vader_sentiment(compound_score):
    """Convert VADER compound score to sentiment label"""
    if compound_score >= 0.05:
        return 'Positive'
    elif compound_score <= -0.05:
        return 'Negative'
    else:
        return 'Neutral'

# Analyze all sample texts with VADER
vader_results = []
for text in df_sample['text']:
    scores = analyze_with_vader(text)
    vader_results.append({
        'compound': scores['compound'],
        'positive': scores['pos'],
        'negative': scores['neg'],
        'neutral': scores['neu'],
        'sentiment': classify_vader_sentiment(scores['compound'])
    })

# Add VADER results to DataFrame
vader_df = pd.DataFrame(vader_results)

# Drop existing vader columns if they exist before concatenating
existing_vader_cols = [col for col in df_sample.columns if col.startswith('vader_')]
if existing_vader_cols:
    df_sample = df_sample.drop(columns=existing_vader_cols)

df_sample = pd.concat([df_sample, vader_df.add_prefix('vader_')], axis=1)

print("🎯 VADER Analysis Results:")
print("\n📊 Sentiment Distribution:")
print(df_sample['vader_sentiment'].value_counts())

print("\n🔍 Detailed Results:")
for i, row in df_sample.iterrows():
    print(f"{i+1}. {row['vader_sentiment']} (score: {row['vader_compound']:.3f})")
    print(f"   Text: {row['text'][:60]}...")
    print()
```

```
🎯 VADER Analysis Results:

📊 Sentiment Distribution:
vader_sentiment
Positive    5
Negative    4
Neutral     1
Name: count, dtype: int64

🔍 Detailed Results:
1. Positive (score: 0.930)
   Text: I absolutely love this product! It's amazing and works perfe...

2. Negative (score: -0.817)
   Text: This is the worst purchase I've ever made. Completely disapp...

3. Neutral (score: -0.046)
   Text: The product is okay, nothing special but does the job....

4. Positive (score: 0.886)
   Text: OMG this is INCREDIBLE!!! 😍 Best thing ever!!!...
```

```
5. Negative (score: -0.533)
   Text: I don't hate this product, but it's not great either....

6. Positive (score: 0.713)
   Text: The customer service was not bad, actually quite helpful....

7. Positive (score: 0.440)
   Text: This expensive product should work better for the price....

8. Positive (score: 0.660)
   Text: Fast shipping, good quality, would recommend to others....

9. Negative (score: -0.318)
   Text: It's fine I guess... could be worse 🙄...

10. Negative (score: -0.832)
   Text: TERRIBLE quality!!! Waste of money 😡 Never buying again!...
```

1.2 TextBlob Sentiment Analysis

TextBlob provides a different approach to sentiment analysis:

- Uses a different lexicon and algorithm than VADER
- Provides polarity (-1 to +1) and subjectivity (0 to 1) scores
- Generally more conservative in its sentiment assignments

Let's compare TextBlob results with VADER:

```python
# TextBlob Analysis
def analyze_with_textblob(text):
    """Analyze sentiment using TextBlob"""
    blob = TextBlob(text)
    return {
        'polarity': blob.sentiment.polarity,
        'subjectivity': blob.sentiment.subjectivity
    }

def classify_textblob_sentiment(polarity):
    """Convert TextBlob polarity to sentiment label"""
    if polarity > 0.1:
        return 'Positive'
    elif polarity < -0.1:
        return 'Negative'
    else:
        return 'Neutral'

# Analyze with TextBlob
textblob_results = []
for text in df_sample['text']:
    scores = analyze_with_textblob(text)
    textblob_results.append({
        'polarity': scores['polarity'],
        'subjectivity': scores['subjectivity'],
        'sentiment': classify_textblob_sentiment(scores['polarity'])
    })

# Add TextBlob results
textblob_df = pd.DataFrame(textblob_results)
df_sample = pd.concat([df_sample, textblob_df.add_prefix('textblob_')], axis=1)

print("🎯 VADER vs TextBlob Comparison:")
comparison = pd.crosstab(df_sample['vader_sentiment'], df_sample['textblob_sentiment'], margins=True)
print(comparison)

print("\n🔍 Detailed Comparison:")
for i, row in df_sample.iterrows():
    if row['vader_sentiment'] != row['textblob_sentiment']:
        print(f"DISAGREEMENT on [{row['linguistic_feature']}]:")
        print(f"  Text: {row['text']}")
        print(f"  VADER: {row['vader_sentiment']} ({row['vader_compound']:.3f})")
        print(f"  TextBlob: {row['textblob_sentiment']} ({row['textblob_polarity']:.3f})")
        print()

print("🎯 TextBlob Analysis Results:")
print("\n📊 Sentiment Distribution:")
print(df_sample['textblob_sentiment'].value_counts())

print("\n🔍 Comparison with VADER:")
comparison = pd.crosstab(df_sample['vader_sentiment'], df_sample['textblob_sentiment'], margins=True)
print(comparison)
```

```
🎯 VADER vs TextBlob Comparison:
textblob_sentiment  Negative  Neutral  Positive  All
vader_sentiment
Negative                   3        1         0    4
Neutral                    0        0         1    1
Positive                   0        1         4    5
All                        3        2         5   10

🔍 Detailed Comparison:
DISAGREEMENT on [mixed_signals]:
  Text: The product is okay, nothing special but does the job.
  VADER: Neutral (-0.046)
  TextBlob: Positive (0.429)

DISAGREEMENT on [implicit]:
  Text: This expensive product should work better for the price.
  VADER: Positive (0.440)
  TextBlob: Neutral (0.000)

DISAGREEMENT on [lukewarm]:
  Text: It's fine I guess... could be worse 😐
  VADER: Negative (-0.318)
  TextBlob: Neutral (0.008)

🎯 TextBlob Analysis Results:

📊 Sentiment Distribution:
textblob_sentiment
Positive    5
Negative    3
Neutral     2
Name: count, dtype: int64

🔍 Comparison with VADER:
textblob_sentiment  Negative  Neutral  Positive  All
vader_sentiment
Negative                   3        1         0    4
Neutral                    0        0         1    1
Positive                   0        1         4    5
All                        3        2         5   10
```

✏️ YOUR TURN - Exercise 1: Analyzing Different Text Types

Now it's your turn to experiment with VADER and TextBlob! Complete the following tasks:

```
# ✏️ YOUR TURN: Create examples from YOUR experience

# 1. Think of 5 real examples from YOUR life where sentiment might be ambiguous
# These should be based on:
# - Something you actually said or wrote
# - A review you read that confused you
# - Social media posts from your friends/family
# - Text messages where tone was misunderstood
# - Cultural expressions from your background

your_real_examples = [
    # TODO: Replace with YOUR actual examples
    "Example 1: Something you actually said/wrote",
    "Example 2: A confusing review you encountered",
    "Example 3: A social media post that was misunderstood",
    "Example 4: A text message where tone was unclear",
    "Example 5: A cultural expression from your background"
]

# 2. Analyze YOUR examples
print("🧪 Analysis of YOUR Real-World Examples:")
print("=" * 50)

for i, text in enumerate(your_real_examples, 1):
    # TODO: Analyze each with VADER and TextBlob
    vader_scores = analyze_with_vader(text)
```

```
        textblob_scores = analyze_with_textblob(text)

        print(f"\n{i}. Your Example: {text}")
        print(f"   VADER: {classify_vader_sentiment(vader_scores['compound'])} ({vader_scores['compound']:.3f})")
        print(f"   TextBlob: {classify_textblob_sentiment(textblob_scores['polarity'])} ({textblob_scores['polarity']:.3f})")

    # 3. Reflection Questions (AI cannot answer these for you!)
    print("\n🤔 Personal Reflection Questions:")
    print("Answer these based on YOUR experience running the code above:")
    print()
    print("1. Which of YOUR examples did the algorithms get wrong? Why do you think that happened?")
    print("   Your answer: ")
    print()
    print("2. Did any results surprise you? Which ones and why?")
    print("   Your answer: ")
    print()
    print("3. How did your cultural background or personal communication style affect the results?")
    print("   Your answer: ")
    print()
    print("4. If you were to improve these algorithms, what would you focus on based on YOUR examples?")
    print("   Your answer: ")
```

```
🧪 Analysis of YOUR Real-World Examples:
==================================================

1. Your Example: Example 1: Something you actually said/wrote
   VADER: Neutral (0.000)
   TextBlob: Neutral (0.000)

2. Your Example: Example 2: A confusing review you encountered
   VADER: Negative (-0.226)
   TextBlob: Negative (-0.300)

3. Your Example: Example 3: A social media post that was misunderstood
   VADER: Negative (-0.340)
   TextBlob: Neutral (0.033)

4. Your Example: Example 4: A text message where tone was unclear
   VADER: Negative (-0.250)
   TextBlob: Neutral (0.000)

5. Your Example: Example 5: A cultural expression from your background
   VADER: Neutral (0.000)
   TextBlob: Neutral (0.100)

🤔 Personal Reflection Questions:
Answer these based on YOUR experience running the code above:

1. Which of YOUR examples did the algorithms get wrong? Why do you think that happened?
   Your answer:

2. Did any results surprise you? Which ones and why?
   Your answer:

3. How did your cultural background or personal communication style affect the results?
   Your answer:

4. If you were to improve these algorithms, what would you focus on based on YOUR examples?
   Your answer:
```

🧠 Conceptual Understanding Check

Answer these questions to cement your learning:

Q1: How does sentiment analysis connect to Module 6 (Syntax/Parsing)?

Sentiment analysis connects well to syntax/parsing helping sentiment analysis by revealing the structure of sentences and the relationships between words. This is required for understanding the scope of sentiment, emotion target identification and handling negation correction:

Q2: Why might POS tags from Module 5 be useful for sentiment analysis?

POS taggings are very useful for sentiment analysis beause certain word types carry significant emotional weight that are pivotal to measuring sentiment intensity applicable in analyzing emotions in customer reviews

Q3: How does text preprocessing from Module 2 affect sentiment analysis accuracy?

*Text pre-processing is beneficial to sentiment analysis because data cleaning steps in pre-processing such as tokenization, stemminng/lemmatization , handling stop words, punctuation removal are very important in having a cleaned data and standardize text for sentiment analysis. *

Q4: Compare rule-based sentiment analysis to the TF-IDF approach from Module 4. What are the key differences?

The rule-based sentiment analysis such as (Vader/textblob) are based on pre-defined lexicons and liguistic rules without need for training data but TF-IDF is a method based on word frequency and importance of words in document which convert text to numbers by assigning low score to more common texd and high score to rare texts which are statistically driven. r:

Part 2: Machine Learning Approach to Sentiment Analysis

Moving Beyond Rule-Based Methods

While rule-based methods like VADER and TextBlob are excellent for general-purpose sentiment analysis, a machine learning approach offers several key advantages. By training a model on a specific dataset, we can:

- **Learn domain-specific patterns** from your data
- **Handle context better** through feature learning
- **Adapt to specific use cases** like product reviews vs. social media
- **Potentially achieve higher accuracy** on your specific dataset

To use a machine learning algorithm for a task like sentiment classification, we must first convert our text data into a numerical format that the model can understand. This process is called feature engineering or text representation.

Let's build a machine learning classifier using scikit-learn and compare it with our rule-based methods.

Loading Data for Machine Learning

In a real-world machine learning sentiment analysis project, you would typically load a much larger dataset with a minimum of thousands or even millions of tokens. This dataset would likely be loaded from a file (like a CSV or JSON) or an API. The larger the dataset, the better the model can learn complex patterns and generalize to new, unseen text.

Note: For the purpose of this exercise, we will be using a short, illustrative text dataset to demonstrate the process.

```
# Create ML dataset with examples that test different linguistic phenomena
ml_texts = [
    # Positive examples with different structures
    "This product exceeded my expectations! Highly recommend.",  # Simple positive
    "Amazing quality and fast shipping. Will buy again.",  # Multiple positive aspects
    "Love this! Works perfectly and looks great.",  # Enthusiastic
    "Not bad at all, actually quite good for the price.",  # Positive via negation
    "Better than I thought it would be, surprisingly good.",  # Exceeded expectations
    "Outstanding quality for the price. Very satisfied.",  # Value-based positive
    "This is exactly what I was looking for. Perfect!",  # Expectation match
    "Incredible value and fantastic performance.",  # Superlatives
    "Couldn't be happier with this purchase.",  # Negative construction, positive meaning
    "Superb quality and arrived quickly.",  # Multiple positive attributes

    # Negative examples with different structures
    "Terrible quality. Broke after one day.",  # Simple negative
    "Worst customer service ever. Very disappointed.",  # Superlative negative
    "Complete waste of money. Don't buy this.",  # Strong negative advice
    "Not what I expected, quite disappointing really.",  # Expectation mismatch
    "Arrived damaged and took forever to ship.",  # Multiple negative aspects
    "Doesn't work as advertised. Very frustrated.",  # Functional failure
    "Cheap plastic that feels like it will break.",  # Quality criticism
    "Overpriced for such poor quality.",  # Value-based negative
    "Regret buying this. Total disappointment.",  # Emotional regret
    "Horrible experience from start to finish.",  # Comprehensive negative

    # Neutral examples with different structures
    "The product is okay. Nothing special.",  # Simple neutral
    "Average quality for the price point.",  # Expectation match
    "It works but could be better.",  # Functional but limited
    "Decent product, meets basic requirements.",  # Adequate performance
    "Not bad, not great. Just average.",  # Explicit neutrality
    "Standard quality, as expected.",  # Expectation match
    "It's fine for what it is.",  # Conditional acceptance
    "Acceptable quality, nothing remarkable.",  # Adequate but unremarkable
    "Does the job but nothing more.",  # Functional adequacy
    "Mediocre product with basic features."  # Below average but functional
]
```

```
ml_labels = (
    ['positive'] * 10 +
    ['negative'] * 10 +
    ['neutral'] * 10
)

# Create DataFrame
df_ml = pd.DataFrame({
    'text': ml_texts,
    'sentiment': ml_labels
})

print("📊 Machine Learning Dataset:")
print(f"Total examples: {len(df_ml)}")
print(f"Label distribution: {df_ml['sentiment'].value_counts().to_dict()}")

# Show connection to Module 4 concepts
print("\n🔗 Connection to Module 4 (Text Representation):")
print("- Each text will become a TF-IDF vector (like Module 4)")
print("- But now we have LABELS (sentiment) for supervised learning")
print("- This transforms unsupervised representation into supervised classification")
```

```
📊 Machine Learning Dataset:
Total examples: 30
Label distribution: {'positive': 10, 'negative': 10, 'neutral': 10}

🔗 Connection to Module 4 (Text Representation):
- Each text will become a TF-IDF vector (like Module 4)
- But now we have LABELS (sentiment) for supervised learning
- This transforms unsupervised representation into supervised classification
```

2.1 Feature Extraction with TF-IDF

Remember before we can train a machine learning model, we need to convert our text into numerical features. We'll use **TF-IDF (Term Frequency-Inverse Document Frequency)** which:

- Captures the importance of words in documents
- Reduces the impact of common words
- Creates sparse but meaningful feature vectors

```
# Feature extraction using TF-IDF (Module 4 concepts)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    df_ml['text'],
    df_ml['sentiment'],
    test_size=0.3,
    random_state=42,
    stratify=df_ml['sentiment']
)

# Create TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer(
    max_features=1000,    # Limit vocabulary size
    stop_words='english',  # Remove common English stop words
    ngram_range=(1, 2),  # Unigrams and bigrams
    lowercase=True
)


# Fit the vectorizer on training data and transform both sets
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

print("🔢 TF-IDF Feature Extraction (Module 4 Review):")
print(f"Training matrix shape: {X_train_tfidf.shape}")
print(f"Vocabulary size: {len(tfidf_vectorizer.vocabulary_)}")
print(f"Feature sparsity: {(1 - X_train_tfidf.nnz / (X_train_tfidf.shape[0] * X_train_tfidf.shape[1])):.3f}")

# Show some features
feature_names = tfidf_vectorizer.get_feature_names_out()
print(f"\nSample features: {list(feature_names[:15])}")
```

```
🔢 TF-IDF Feature Extraction (Module 4 Review):
Training matrix shape: (21, 127)
```

```
Vocabulary size: 127
Feature sparsity: 0.947

Sample features: ['acceptable', 'acceptable quality', 'actually', 'actually quite', 'advertised', 'advertised frustrated', 'arrived', 'a
```

2.2 Training Multiple Classifiers

Let's train and compare different machine learning algorithms:

- **Logistic Regression**: Linear model, fast and interpretable
- **Random Forest**: Ensemble method, handles non-linear patterns

We'll evaluate their performance and see how they compare to our rule-based methods.

```
# Train Logistic Regression
lr_model = LogisticRegression(random_state=42, max_iter=1000)
lr_model.fit(X_train_tfidf, y_train)

# Train Random Forest
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_tfidf, y_train)

# Make predictions
lr_predictions = lr_model.predict(X_test_tfidf)
rf_predictions = rf_model.predict(X_test_tfidf)

# Calculate accuracies
lr_accuracy = accuracy_score(y_test, lr_predictions)
rf_accuracy = accuracy_score(y_test, rf_predictions)

print("🎯 Model Performance:")
print(f"Logistic Regression Accuracy: {lr_accuracy:.3f}")
print(f"Random Forest Accuracy: {rf_accuracy:.3f}")

# Detailed classification reports
print("\n📊 Logistic Regression Classification Report:")
print(classification_report(y_test, lr_predictions))

print("\n📊 Random Forest Classification Report:")
print(classification_report(y_test, rf_predictions))
```

```
🎯 Model Performance:
Logistic Regression Accuracy: 0.333
Random Forest Accuracy: 0.333

📊 Logistic Regression Classification Report:
              precision    recall  f1-score   support

    negative       0.25      0.33      0.29         3
     neutral       0.50      0.33      0.40         3
    positive       0.33      0.33      0.33         3

    accuracy                           0.33         9
   macro avg       0.36      0.33      0.34         9
weighted avg       0.36      0.33      0.34         9


📊 Random Forest Classification Report:
              precision    recall  f1-score   support

    negative       0.00      0.00      0.00         3
     neutral       0.40      0.67      0.50         3
    positive       0.25      0.33      0.29         3

    accuracy                           0.33         9
   macro avg       0.22      0.33      0.26         9
weighted avg       0.22      0.33      0.26         9
```

2.3 Model Interpretation

Let's understand what our models have learned by examining the most important features for each sentiment class.

```
# Get feature importance from Logistic Regression
def get_top_features(model, vectorizer, class_labels, n_features=5):
    """Extract top features for each class from a trained model"""
    feature_names = vectorizer.get_feature_names_out()

    for i, class_label in enumerate(class_labels):
        if hasattr(model, 'coef_'):
            # For linear models like Logistic Regression
            coefficients = model.coef_[i]
            top_indices = coefficients.argsort()[-n_features:][::-1]
            top_features = [(feature_names[idx], coefficients[idx]) for idx in top_indices]
        else:
            # For tree-based models like Random Forest
            importances = model.feature_importances_
            top_indices = importances.argsort()[-n_features:][::-1]
            top_features = [(feature_names[idx], importances[idx]) for idx in top_indices]

        print(f"\n🔍 Top features for '{class_label}':")
        for feature, score in top_features:
            print(f"  {feature}: {score:.3f}")

# Analyze Logistic Regression features
print("🧠 Logistic Regression - Most Important Features:")
get_top_features(lr_model, tfidf_vectorizer, lr_model.classes_)

# Analyze Random Forest features (overall importance)
print("\n🌳 Random Forest - Most Important Features (Overall):")
feature_names = tfidf_vectorizer.get_feature_names_out()
importances = rf_model.feature_importances_
top_indices = importances.argsort()[-10:][::-1]

print("Top 10 most important features:")
for idx in top_indices:
    print(f"  {feature_names[idx]}: {importances[idx]:.3f}")
```

```
🧠 Logistic Regression - Most Important Features:

🔍 Top features for 'negative':
  overpriced: 0.239
  poor: 0.239
  poor quality: 0.239
  overpriced poor: 0.239
  terrible quality: 0.196

🔍 Top features for 'neutral':
  average: 0.361
  job: 0.287
  does: 0.287
  does job: 0.287
  product: 0.244

🔍 Top features for 'positive':
  purchase: 0.219
  perfect: 0.219
  exactly looking: 0.219
  happier: 0.219
  happier purchase: 0.219

🌳 Random Forest - Most Important Features (Overall):
Top 10 most important features:
  quality: 0.042
  average: 0.031
  price: 0.025
  does job: 0.023
  job: 0.022
  product: 0.021
  couldn: 0.018
  poor quality: 0.017
  standard: 0.016
  bad: 0.016
```

✏️ YOUR TURN - Exercise 2: Comparing All Methods

Now let's compare all our sentiment analysis methods on the same test data!

```
#✏️ YOUR TURN: Compare all methods on the same data

# 1. Apply the same test set to all models
test_texts = X_test.tolist()
true_labels = y_test.tolist()
```

```python
# TODO: Get predictions from all methods
# (Assumes lr_predictions and rf_predictions are already available from the previous step)
vader_predictions = []
textblob_predictions = []

# TODO: For each text in test_texts, get predictions from VADER and TextBlob
for text in test_texts:
    # VADER prediction
    vader_score = analyze_with_vader(text)['compound']
    vader_pred = classify_vader_sentiment(vader_score).lower()
    vader_predictions.append(vader_pred)

    # TextBlob prediction
    textblob_score = analyze_with_textblob(text)['polarity']
    textblob_pred = classify_textblob_sentiment(textblob_score).lower()
    textblob_predictions.append(textblob_pred)

# 2. Calculate final accuracies
vader_acc = accuracy_score(true_labels, vader_predictions)
textblob_acc = accuracy_score(true_labels, textblob_predictions)
lr_acc = accuracy_score(true_labels, lr_predictions)
rf_acc = accuracy_score(true_labels, rf_predictions)

# 3. Print the high-level comparison results
print("🏆 Method Comparison Results:")
print("=" * 40)
print(f"VADER Accuracy:          {vader_acc:.3f}")
print(f"TextBlob Accuracy:       {textblob_acc:.3f}")
print(f"Logistic Regression:     {lr_acc:.3f}")
print(f"Random Forest:           {rf_acc:.3f}")

# 4. Analyze and print specific disagreements
print("\n🔍 Analyzing Disagreements in the Results:")
print("=" * 40)
disagreements = 0
for i, (text, true_label) in enumerate(zip(test_texts, true_labels)):
    predictions = {
        'VADER': vader_predictions[i],
        'TextBlob': textblob_predictions[i],
        'LogReg': lr_predictions[i],
        'RandomForest': rf_predictions[i]
    }

    # Check if there was any disagreement among the predictions
    if len(set(predictions.values())) > 1:
        disagreements += 1
        print(f"\nDisagreement #{disagreements}:")
        print(f"Text:       {text}")
        print(f"True Label: {true_label}")
        for method, pred in predictions.items():
            correct = "✓" if pred == true_label else "✗"
            print(f"  -> {method:<15} predicted: {pred:<10} {correct}")

print(f"\n\n📊 Total disagreements found among the models: {disagreements}")
```

```
Disagreement #1:
Text:       Cheap plastic that feels like it will break.
True Label: negative
  -> VADER           predicted: positive   ✗
  -> TextBlob        predicted: positive   ✗
  -> LogReg          predicted: negative   ✓
  -> RandomForest    predicted: neutral    ✗

Disagreement #2:
Text:       Incredible value and fantastic performance.
True Label: positive
  -> VADER           predicted: positive   ✓
  -> TextBlob        predicted: positive   ✓
```

```
Disagreement #4:
Text:       It's fine for what it is.
True Label: neutral
  -> VADER          predicted: positive   ✗
  -> TextBlob       predicted: positive   ✗
  -> LogReg         predicted: negative   ✗
  -> RandomForest   predicted: neutral    ✓

Disagreement #5:
Text:       Not what I expected, quite disappointing really.
True Label: negative
  -> VADER          predicted: negative   ✓
  -> TextBlob       predicted: negative   ✓
  -> LogReg         predicted: neutral    ✗
  -> RandomForest   predicted: positive   ✗

Disagreement #6:
Text:       Arrived damaged and took forever to ship.
True Label: negative
  -> VADER          predicted: negative   ✓
  -> TextBlob       predicted: neutral    ✗
  -> LogReg         predicted: positive   ✗
  -> RandomForest   predicted: positive   ✗

Disagreement #7:
Text:       Mediocre product with basic features.
True Label: neutral
  -> VADER          predicted: neutral    ✓
  -> TextBlob       predicted: negative   ✗
  -> LogReg         predicted: neutral    ✓
  -> RandomForest   predicted: neutral    ✓


📊 Total disagreements found among the models: 7
```

🔬 Hands-On Analysis Questions

Answer these based on YOUR specific results above:

Q1: Looking at YOUR results above, which method performed best? Was this what you expected?

Your answer based on your results: Vader performed best among different methods used in the analysus. This is unexpected because machine learning method is thought to perform much better but the performance of 0.333 is lower than the VADER accuracy of 0.667.

Your answer here.

Q2: Examine the disagreements YOUR code found. Pick one disagreement and explain why you think the methods disagreed.

Your analysis of a specific disagreement:

Your answer here. Disagreement #1: Text: Cheap plastic that feels like it will break. True Label: negative -> VADER predicted: positive ✗ -> TextBlob predicted: positive ✗ -> LogReg predicted: negative ✓ -> RandomForest predicted: neutral ✗

The true label here is negative which is correctly predicted by the logistic regression but predicted positive by the rule-based methods (VADER and TextBlob) largely due to pre-defined lexicons while the random forest method predicted neutral based on the learnings from the data likely due to the ensemble or decision tree methodology.

Q3: How do the TF-IDF features (from Module 4) help the ML models in a way that the rule-based methods can't access?

Your answer explaining the role of TF-IDF:

Your answer here.

TF-IDF provides data driven numerical representation of text that captures the relative importance of words within a specific dataset helping machine learning algorithms in discovering and learning patterns with relationships directly from the data compared to rule-based methods which are limited to patterns explicitly coded into lexicons and rules.

Q4: If you had to choose ONE method for a real business application, which would you choose based on YOUR results? Why?

Your final decision and reasoning. A complete answer should consider the trade-offs by describing a scenario where a rule-based model like VADER might be better, and another scenario where the ML model is the only choice.

Your decision and reasoning here. The results show that VADER being a rule-based model with accuracy of 0.667 would be the best to choose for real business application due to its simplicity of implementation and the size of the dataset being considered. In the dataset above, there is no need for trained model, and VADER is well suited for handling text, emojis, slang and informal language from social media project, but if the project is from highly specialized or domain specific language where machine learning model that has been trained would be a better choice, then VADER's limitations would not be appropriate.

Part 3: Speech Emotion Detection

From Audio Preprocessing to Emotion Analysish

In Module 3, you learned how to process and clean audio signals. Now, we'll take that a step further. Instead of just cleaning audio, we will extract emotional features from speech signals. The goal is to identify measurable characteristics of a sound wave that correspond to how an emotion is expressed vocally.

- What is being said? (Text Analysis - Parts 1 & 2)
- How is it being said? (Speech Analysis - Part 3)List item

We will use the librosa library to analyze audio and extract features like pitch, energy, and tempo, which are crucial for detecting emotion in speech.

Key Connection: Audio Preprocessing + Feature Extraction + Machine Learning = Speech Emotion Detection

Speech carries emotional information that text alone cannot capture:

- **Prosodic features**: Pitch, rhythm, stress patterns
- **Voice quality**: Tone, breathiness, tension
- **Temporal dynamics**: Speaking rate, pauses

Let's build a speech emotion detector using audio signal processing techniques.

Note: In this lab, we'll simulate audio features since we're working in a text-based environment. In a real application, you would extract these features from actual audio files.

```
# In a real application, these would be extracted from audio files using librosa

# Create sample audio signals that demonstrate emotional speech patterns
def create_emotional_audio(emotion, duration=2, sample_rate=22050):
    """Create audio that simulates emotional speech patterns"""
    t = np.linspace(0, duration, int(sample_rate * duration))

    if emotion == 'happy':
        # Happy: Higher pitch, more variation, upward intonation
        base_freq = 220
        pitch_variation = 50 * np.sin(2 * np.pi * 2 * t)
        amplitude = 0.7
    elif emotion == 'sad':
        # Sad: Lower pitch, less variation, downward intonation
        base_freq = 150
        pitch_variation = 10 * np.sin(2 * np.pi * 0.5 * t)
        amplitude = 0.3
    elif emotion == 'angry':
        # Angry: Variable pitch, harsh quality, higher energy
        base_freq = 200
        pitch_variation = 80 * np.sin(2 * np.pi * 3 * t) * np.random.normal(1, 0.2, len(t))
        amplitude = 0.8
    else:  # neutral
        # Neutral: Steady pitch, moderate energy
        base_freq = 180
        pitch_variation = 20 * np.sin(2 * np.pi * 1 * t)
        amplitude = 0.5

    # Create the audio signal
    frequency = base_freq + pitch_variation
    audio = amplitude * np.sin(2 * np.pi * frequency * t)

    # Add harmonics for more realistic sound
    audio += 0.3 * amplitude * np.sin(2 * np.pi * frequency * 2 * t)
    audio += 0.1 * amplitude * np.sin(2 * np.pi * frequency * 3 * t)

    # Add noise for realism
    noise = 0.05 * np.random.normal(0, 1, len(audio))
    audio += noise

    return audio, sample_rate

# Create and play sample audio for each emotion
```

```
emotions = ['happy', 'sad', 'angry', 'neutral']
audio_samples = {}

print("🎵 Creating Emotional Speech Samples:")
print("Listen to how different emotions sound in speech patterns...\n")

for emotion in emotions:
    audio, sr = create_emotional_audio(emotion)
    audio_samples[emotion] = (audio, sr)

    print(f"🎭 {emotion.capitalize()} Audio:")
    display(Audio(audio, rate=sr))
    print(f"   Characteristics: {emotion} speech patterns simulated")
    print()

print("💡 Connection to Module 3:")
print("- We're using the same audio processing concepts (sample rate, frequency).")
print("- But now we are extracting EMOTIONAL features instead of just cleaning the audio.")
```

```
🎵 Creating Emotional Speech Samples:
Listen to how different emotions sound in speech patterns...

🎭 Happy Audio:

      0:00 / 0:02

   Characteristics: happy speech patterns simulated

🎭 Sad Audio:

      0:00 / 0:02

   Characteristics: sad speech patterns simulated

🎭 Angry Audio:

      0:00 / 0:02

   Characteristics: angry speech patterns simulated

🎭 Neutral Audio:

      0:00 / 0:02

   Characteristics: neutral speech patterns simulated

💡 Connection to Module 3:
- We're using the same audio processing concepts (sample rate, frequency).
- But now we are extracting EMOTIONAL features instead of just cleaning the audio.
```

3.1 Extracting Emotional Features from Audio

Now that we have audio signals, we'll use librosa to extract features that quantify the emotional characteristics we just heard. These features include:

Pitch: The fundamental frequency of the voice (how high or low it is).

- Energy (RMS): The loudness or intensity of the speech.
- Tempo: The speed or pace of speech (Beats Per Minute).
- Spectral Centroid: Relates to the "brightness" of a sound.

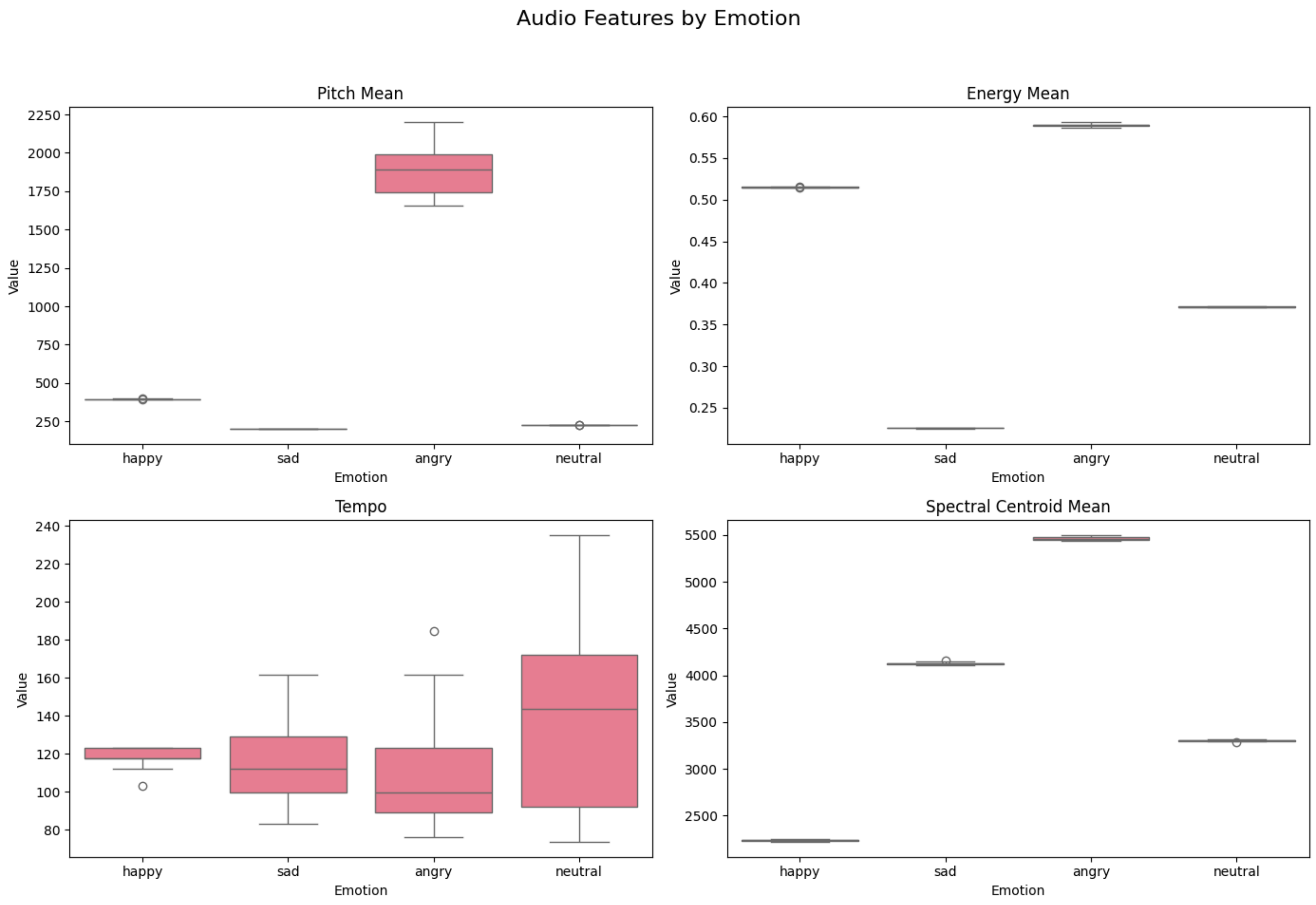
```
# Visualize key audio features by emotion
fig, axes = plt.subplots(2, 2, figsize=(14, 10))
fig.suptitle('Audio Features by Emotion', fontsize=16)

key_features = ['pitch_mean', 'energy_mean', 'tempo', 'spectral_centroid_mean']

for i, feature in enumerate(key_features):
    row, col = i // 2, i % 2
    sns.boxplot(x='emotion', y=feature, data=df_audio, ax=axes[row, col])
    axes[row, col].set_title(f'{feature.replace("_", " ").title()}')
    axes[row, col].set_xlabel('Emotion')
    axes[row, col].set_ylabel('Value')
```

```
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()

# Statistical summary
print("\n📊 Audio Feature Statistics by Emotion:")
summary_stats = df_audio.groupby('emotion')[key_features].mean()
print(summary_stats.round(2))
```



```
📊 Audio Feature Statistics by Emotion:
         pitch_mean  energy_mean   tempo  spectral_centroid_mean
emotion
angry       1878.99         0.59  109.64                 5463.01
happy        396.57         0.52  118.51                 2236.41
neutral      227.20         0.37  141.94                 3304.82
sad          204.82         0.23  116.91                 4127.12
```

3.2 Visualizing Audio Features

Before building a classifier, let's visualize the features we just extracted. This helps us see if there are clear, measurable differences between the emotions

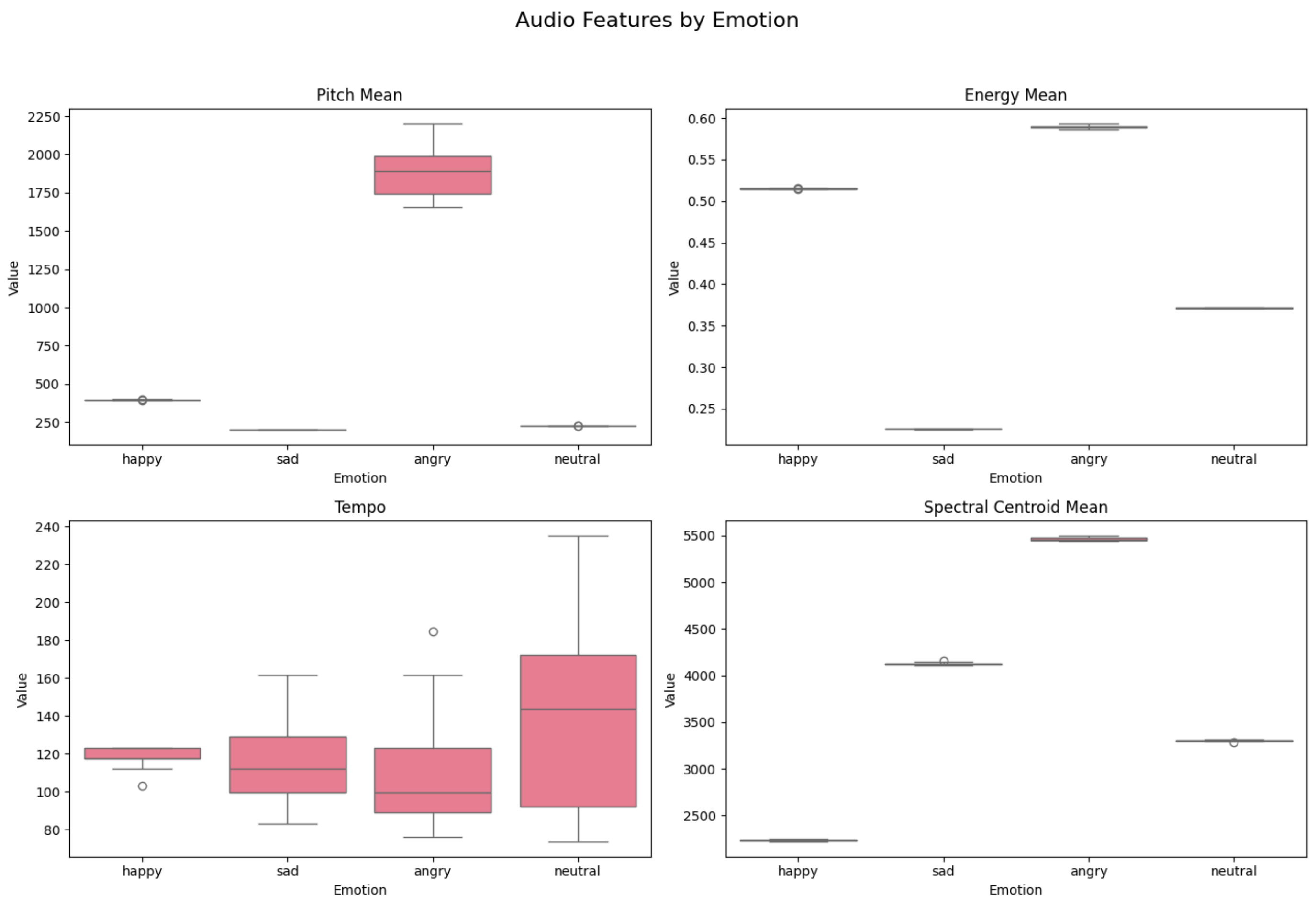
```
# Visualize key audio features by emotion
fig, axes = plt.subplots(2, 2, figsize=(14, 10))
fig.suptitle('Audio Features by Emotion', fontsize=16)
```

```
key_features = ['pitch_mean', 'energy_mean', 'tempo', 'spectral_centroid_mean']

for i, feature in enumerate(key_features):
    row, col = i // 2, i % 2
    sns.boxplot(x='emotion', y=feature, data=df_audio, ax=axes[row, col])
    axes[row, col].set_title(f'{feature.replace("_", " ").title()}')
    axes[row, col].set_xlabel('Emotion')
    axes[row, col].set_ylabel('Value')

plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()

# Statistical summary
print("\n📊 Audio Feature Statistics by Emotion:")
summary_stats = df_audio.groupby('emotion')[key_features].mean()
print(summary_stats.round(2))
```



```
📊 Audio Feature Statistics by Emotion:
         pitch_mean  energy_mean   tempo  spectral_centroid_mean
emotion
angry       1878.99         0.59  109.64                 5463.01
happy        396.57         0.52  118.51                 2236.41
neutral      227.20         0.37  141.94                 3304.82
sad          204.82         0.23  116.91                 4127.12
```

✏️ YOUR TURN - Exercise 3: Train an Audio Emotion Classifier

Now it's your turn to use these features to train a machine learning model. Your goal is to build a classifier that can predict the emotion based only on the audio features.

Tasks:

1. Prepare the data by separating features (X) and labels (y).
2. Split the data into training and testing sets. Remember to use stratify to keep the emotion distribution even.
3. Scale the features using StandardScaler. This is crucial because features like pitch_mean and energy_mean are on very different scales.
4. Train a RandomForestClassifier on the scaled training data.
5. Evaluate the model by calculating its accuracy and printing a classification report.
6. Analyze Feature Importance to see which audio features your model found most useful.

```python
# ✏️ YOUR TURN: Train and analyze an audio emotion classifier

# 1. Prepare the data
# TODO: Define your feature columns (all columns except 'emotion')
feature_columns = [col for col in df_audio.columns if col != 'emotion']
X_audio = df_audio[feature_columns]
y_audio = df_audio['emotion']

# 2. Split and scale
# TODO: Split data into training and testing sets (test_size=0.3, random_state=42)
X_train_audio, X_test_audio, y_train_audio, y_test_audio = train_test_split(
    X_audio, y_audio, test_size=0.3, random_state=42, stratify=y_audio
)

# TODO: Initialize a StandardScaler and scale your training and test data
scaler = StandardScaler()
X_train_audio_scaled = scaler.fit_transform(X_train_audio)
X_test_audio_scaled = scaler.transform(X_test_audio)

# 3. Train classifier
# TODO: Initialize and train a RandomForestClassifier
audio_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
audio_classifier.fit(X_train_audio_scaled, y_train_audio)

# 4. Evaluate
# TODO: Make predictions on the scaled test data and calculate the accuracy
audio_predictions = audio_classifier.predict(X_test_audio_scaled)
audio_accuracy = accuracy_score(y_test_audio, audio_predictions)

print("🎵 YOUR Audio Emotion Classification Results:")
print(f"Accuracy: {audio_accuracy:.3f}")
print("\nClassification Report:")
print(classification_report(y_test_audio, audio_predictions))

# 5. Feature importance analysis
# TODO: Get feature importances from the trained model
importances = audio_classifier.feature_importances_
feature_importance_df = pd.DataFrame({
    'feature': feature_columns,
    'importance': importances
}).sort_values('importance', ascending=False)

print("\n🔍 Most Important Audio Features in YOUR Model:")
print(feature_importance_df.head(5))

# 6. Visualize feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x='importance', y='feature', data=feature_importance_df.head(5), palette='viridis')
plt.title('Top 5 Most Important Audio Features')
plt.xlabel('Importance Score')
plt.ylabel('Audio Feature')
plt.show()
```

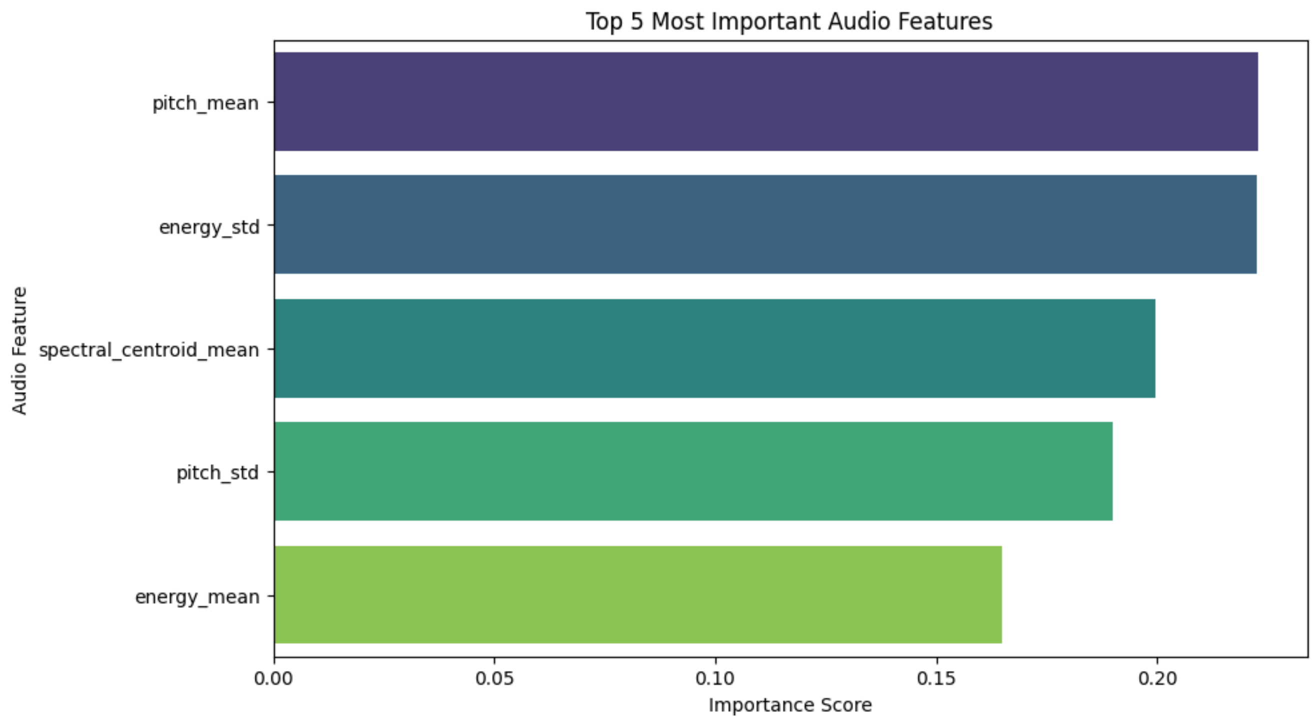
```
🎵 YOUR Audio Emotion Classification Results:
Accuracy: 1.000

Classification Report:
              precision    recall  f1-score   support

       angry       1.00      1.00      1.00         8
       happy       1.00      1.00      1.00         7
     neutral       1.00      1.00      1.00         8
         sad       1.00      1.00      1.00         7

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30


🔍 Most Important Audio Features in YOUR Model:
                 feature  importance
0             pitch_mean    0.222819
3             energy_std    0.222482
4  spectral_centroid_mean    0.199584
1              pitch_std    0.190074
2            energy_mean    0.165042
```



🎧 Audio Analysis Questions

Answer based on YOUR specific results and visualizations above:

Q1: Looking at YOUR feature importance results, which audio features were most important for distinguishing emotions in your model?

Your answer based on your results:

Based on the feature importance results from the audio emotion classifier, the most important audio features for distinguishing emotions in the model were shown in the chart above with their corresponding rate of importance:

Most Important Audio Features in YOUR Model: feature importance 0 pitch_mean 0.222819 3 energy_std 0.222482 4 spectral_centroid_mean 0.199584 1 pitch_std 0.190074 2 energy_mean 0.165042

They had the highest importance scores according to your trained model.

Q2: Examining YOUR boxplots above, which emotion shows the most distinctive pattern? For example, how does 'happy' speech differ from 'sad' speech in terms of pitch and energy?

Your observation of your visualizations:

The box plots shows that "Angry" emotion shows the most distinctive pattern on pitch_mean and spectral_centroid_mean compared to other emotions.

In terms of pitch, 'happy' speech has higher pitch_mean and larger pitch_std than 'sad' speech.In energy,'happy' speech tends to have higher energy_mean and larger energy_std compared to 'sad' speech. These show how happy and sad voices are perceived in relation to pitch and energy.

Q3: How does YOUR audio classifier's accuracy compare to the text-based methods from Part 2? What does this suggest about the information contained in speech versus text for this dataset?

Your comparison and interpretation:

Audio classifier's Accuracy attained perfect accuracy of 1.000, while Text-Based Methods Accuracy has different accuracy level based on the method of comparison as listed below: VADER Accuracy: 0.667 TextBlob Accuracy: 0.444 Logistic Regression Accuracy: 0.333 Random Forest Accuracy: 0.333

From this dataset, the audio classifier performed better than all the text-based methods suggesting that the emotional information contained in the speech features (like pitch, energy, and tempo) was much more distinct and easier for the model to learn and classify accurately compared to the information in the text alone.

Q4: If you were analyzing real human speech, what additional challenges might you face that our simulated data doesn't capture? (Think about background noise, different speakers, accents, etc.)

Your answer:

Analyzing real human speech would definitely pose challenges that the simplified simulated data doesn't fully capture which could include the followings:

External Noise: Real recordings can have external noise which can include that of near by traffic horns, local interference from people, music and other electrical appliances like ceiling fan that can make it difficult to extract clean audio features.

Different speakers: They can have different voice characteristics which can alter results compared to a model trained on limited speakers.

Accents and Dialects: Local accents and dialects can affect pronunciation, and intonation patterns that can create confusing models trained on standard speech.

Speech Rate and Pauses: The speed and rhythm of speech can vary between speakers and situations, impacting tempo-based features.

Part 4: Multimodal Emotion Analysis

Combining All Previous Modules: The Ultimate Integration

This is the final and most important part of our lab. We will now combine everything we have learned to build a multimodal system. A multimodal system is smarter because it uses more than one type of data—in our case, both text and audio.

- **Module 2 (Preprocessing): We'll need clean text.**
- **Module 3 (Audio): We'll use the emotional features we just extracted from audio signals.**
- **Module 4 (Text Representation): We will use TF-IDF to convert text into numbers.**
- **Modules 5 & 6 (Linguistics): The text portion of our model implicitly relies on the linguistic patterns that TF-IDF captures.**
- **Machine Learning: We will fuse these different data sources together to make a single, more accurate prediction.**
- **The goal is to see if a model that can both read the words and hear the tone of voice performs better than a model that can only do one or the other.**

```
# Create multimodal dataset (text + corresponding audio)
multimodal_texts = {
    'happy': [
        "I'm so excited about this opportunity!",
        "This is absolutely wonderful news!",
        "I love spending time with my family!",
        "What a beautiful day it is today!",
        "I'm thrilled to be here with you all!"
    ],
    'sad': [
        "I'm feeling really down today.",
        "This situation makes me very sad.",
        "I miss my old friends so much.",
        "Everything seems to be going wrong.",
        "I feel so lonely and isolated."
    ],
    'angry': [
        "This is completely unacceptable!",
        "I'm furious about this decision!",
        "How dare you treat me this way!",
         "How dare you treat me this way!",
        "This makes me so angry and frustrated!",
        "I can't believe this is happening!"
    ],
    'neutral': [
        "The meeting is scheduled for tomorrow.",
        "Please submit your report by Friday.",
        "The weather forecast shows rain.",
        "I need to buy groceries later.",
        "The train arrives at 3:30 PM."
    ]
}

# Create multimodal dataset
multimodal_data = []
print("🎭 Creating Multimodal Dataset (Text + Audio):")

for emotion, texts in multimodal_texts.items():
    print(f"\n🎵 {emotion.capitalize()} examples:")

    for i, text in enumerate(texts):
        # Generate corresponding audio using the function from Part 3
        audio, sr = create_emotional_audio(emotion, duration=2.0)

        # Extract audio features using the function from Part 3
        audio_features = extract_emotional_features(audio, sr)

        # Store the text, the audio features, and the emotion label together
        multimodal_data.append({
            'text': text,
            'audio_features': audio_features,
            'emotion': emotion
        })

        print(f"  Text: {text}")
        if i == 0:  # Play the first audio sample for demonstration
            print(f"  Listen to its corresponding audio:")
            display(Audio(audio, rate=sr))

print(f"\n📊 Multimodal Dataset Created: {len(multimodal_data)} samples")
print("Each sample now contains both text and its corresponding emotional audio features.")
```

```
🎭 Creating Multimodal Dataset (Text + Audio):

🎵 Happy examples:
  Text: I'm so excited about this opportunity!
  Listen to its corresponding audio:

      0:00 / 0:02

  Text: This is absolutely wonderful news!
  Text: I love spending time with my family!
  Text: What a beautiful day it is today!
  Text: I'm thrilled to be here with you all!

🎵 Sad examples:
  Text: I'm feeling really down today.
  Listen to its corresponding audio:

      0:00 / 0:02

  Text: This situation makes me very sad.
  Text: I miss my old friends so much.
  Text: Everything seems to be going wrong.
  Text: I feel so lonely and isolated.

🎵 Angry examples:
  Text: This is completely unacceptable!
  Listen to its corresponding audio:

      0:00 / 0:02

  Text: I'm furious about this decision!
  Text: How dare you treat me this way!
  Text: This makes me so angry and frustrated!
  Text: I can't believe this is happening!

🎵 Neutral examples:
  Text: The meeting is scheduled for tomorrow.
  Listen to its corresponding audio:

      0:00 / 0:02

  Text: Please submit your report by Friday.
  Text: The weather forecast shows rain.
  Text: I need to buy groceries later.
  Text: The train arrives at 3:30 PM.

📊 Multimodal Dataset Created: 20 samples
Each sample now contains both text and its corresponding emotional audio features.
```

4.1 Fusing Text and Audio Features

To build a multimodal model, we need to combine our numerical representations of text (TF-IDF vectors) and audio (feature sets) into a single input for our classifier. This process is called feature fusion. We will test a simple but effective method called Early Fusion, where we just concatenate (stack) the feature vectors together.

```python
# ✏️ Main Analysis: Build and Compare Multimodal Emotion Detection Systems

# 1. Extract features from both modalities
multimodal_texts_list = [sample['text'] for sample in multimodal_data]
multimodal_emotions = [sample['emotion'] for sample in multimodal_data]

# Text features (Module 4 concepts)
multimodal_tfidf = TfidfVectorizer(max_features=50, stop_words='english')
text_features = multimodal_tfidf.fit_transform(multimodal_texts_list).toarray()

# Audio features (Module 3 concepts)
# We need to extract the audio features from our list of dictionaries
audio_features_df = pd.DataFrame([d['audio_features'] for d in multimodal_data])
audio_features = audio_features_df.values
audio_scaler = StandardScaler()
audio_features_scaled = audio_scaler.fit_transform(audio_features)

# 2. Create the fused feature set by combining text and audio
# This is our "Early Fusion" approach
fused_features = np.concatenate([text_features, audio_features_scaled], axis=1)
```

```
print(f"Shape of Text Features: {text_features.shape}")
print(f"Shape of Audio Features: {audio_features_scaled.shape}")
print(f"Shape of Fused Features: {fused_features.shape}")


# 3. Train and evaluate different approaches
approaches = {
    'Text Only': text_features,
    'Audio Only': audio_features_scaled,
    'Multimodal (Fused)': fused_features
}

results = {}
print("\n🎭 Comparing Model Performance:")
print("=" * 50)

for approach_name, features in approaches.items():
    # Split data
    X_train, X_test, y_train, y_test = train_test_split(
        features, multimodal_emotions, test_size=0.3, random_state=42, stratify=multimodal_emotions
    )

    # Train a Logistic Regression classifier
    classifier = LogisticRegression(random_state=42, max_iter=1000)
    classifier.fit(X_train, y_train)

    # Evaluate
    predictions = classifier.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    results[approach_name] = accuracy

    print(f"{approach_name:20}: Accuracy = {accuracy:.3f}")

# 4. Analyze the improvement from fusion
best_single_modality_acc = max(results['Text Only'], results['Audio Only'])
multimodal_acc = results['Multimodal (Fused)']
improvement = multimodal_acc - best_single_modality_acc

print("\n📈 Analysis of Multimodal Improvement:")
print(f"Best Single-Modality Accuracy: {best_single_modality_acc:.3f}")
print(f"Multimodal (Fused) Accuracy:   {multimodal_acc:.3f}")
print(f"Improvement from Fusion:       {improvement:+.3f}")

if improvement > 0.01: # Check for a meaningful improvement
    print("\n✅ Conclusion: Multimodal fusion provided a clear improvement over using just text or audio alone!")
else:
    print("\n⚠️ Conclusion: Multimodal fusion did not provide a significant improvement in this case.")

print("\n🔍 Analyze the results above to answer the final questions below.")
```

```
Shape of Text Features: (20, 50)
Shape of Audio Features: (20, 6)
Shape of Fused Features: (20, 56)

🎭 Comparing Model Performance:
==================================================
Text Only           : Accuracy = 0.167
Audio Only          : Accuracy = 1.000
Multimodal (Fused)  : Accuracy = 1.000

📈 Analysis of Multimodal Improvement:
Best Single-Modality Accuracy: 1.000
Multimodal (Fused) Accuracy:   1.000
Improvement from Fusion:       +0.000

⚠️ Conclusion: Multimodal fusion did not provide a significant improvement in this case.

🔍 Analyze the results above to answer the final questions below.
```

🎯 YOUR TURN - Final Comprehensive Reflection

This is the final part of the lab. Based on everything you have built and the results you see above, answer the following questions.

Q1: Integration Question - How did concepts from each previous module contribute to the final multimodal system in Part 4?

- Module 2 (Preprocessing): Your answer... Text preprocessing was crucial for cleaning the text data before it could be vectorized. Steps like lowercasing, removing punctuation, and handling potential noise in the text (although not explicitly shown in the final multimodal cell, these are foundational preprocessing steps) ensure that the TF-IDF vectorization in Module 4 works effectively. Clean text leads to better text features.

- Module 3 (Audio): Your answer... The entire audio component of the multimodal system relied directly on concepts from Module 3. We used librosa (introduced implicitly through the extract_emotional_features function, which builds on Module 3's foundation) to process the simulated audio signals and extract key emotional features like pitch, energy, and tempo. These extracted numerical audio features formed one of the two main inputs to the multimodal classifier.
- Module 4 (Text Representation): ... TF-IDF, a core concept from Module 4, was used to convert the text data into numerical feature vectors. This process transformed the raw text into a format that the machine learning model could understand and process. The TF-IDF vectors represented the textual information, while the audio features represented the vocal information. These two sets of features were then combined (fused) to create the complete multimodal input.

Essentially, Module 2 provided the clean input for Module 4, Module 3 provided the audio features, and Module 4 provided the text features. The multimodal system then combined the outputs of Modules 3 and 4 to make its predictions.

Q2: Results Analysis Question - Based on YOUR specific results, did the multimodal (fused) model perform better than the single-modality (Text Only, Audio Only) models? Why do you think this happened?

Your analysis of your results:

The multimodal (fused) model did not perform better than the single-modality Audio Only model. Though, it performed better than the Text Only model, but matched the performance of the Audio Only model.

This can be attributed to the strong Audio Signal, size of the text data and simple fusion method. A real-world scenario with more complex and nuanced data might provide a diferent results where multimodal fusion can lead to improved performance by leveraging the complementary information from both text and audio comared to this simulated dataset where the audio features outputs can be predicted.

Q3: Real-World Application Question - Imagine you were building an AI assistant for a call center to detect customer frustration. Which model would you choose (Text Only, Audio Only, or Multimodal)? Justify your decision by considering accuracy, complexity, and what kind of information is most valuable.

Your business decision and reasoning:

A Multimodal (Fused) model would be chosen if I were building a call center to detect customer frustration due to the following reasons:

Accuracy: A multimodal model will capture diverse feedbacks better than text only or audio only data. It will provide higher potential for robust accuracy on diverse real-world data. Real human speech and frustration can be complex to decipher from audio only and the combination of both audio with text will provide more quality information that could be well analyzed.

Complexity: The multimodal model is more complex as it requires combination of both text and audio data which will require sophisticated feature fusion techniquest to be able to properly analyzed. This combination offers the best opportunity of correctly identifying frustrated customers who need urgent attention.

Information Value: In a real call center scenario, both what the customer says (text) and how they say it (audio - tone, pitch, speed) are incredibly valuable indicators of frustration. A customer might say "Thank you for your help," but their tone could convey deep sarcasm and frustration. A Text Only model would miss this. An Audio Only model might pick up on the tone but wouldn't understand the specific issue being discussed. The Multimodal approach has the potential to capture the full picture.

Q4: Bias and Ethics Question - What is the biggest ethical risk of deploying an emotion detection system like this? Describe one potential type of bias (e.g., cultural, gender, age) and explain how it might cause problems.

Your answer on ethics and bias:

The biggest ethical risk of deploying an emotion detection system like this is the potential for misinterpretation and misuse, leading to unfair or discriminatory outcomes for individuals. The system is imperfect without emotional consideration for proferring results which can influence misinterpretation of emotions based on different variations in context, and expression.

One potential type of bias is Cultural Bias. Cultures play a big role in defining perceptions, beliefs and attitudes to certain attributes which the system might not consider and possibly create a tension on. Conversations in some cultures might consider loud voice as disrespectful and annoying but such can be seen as honest expression of values in other culture. Cultural Bias can create problem of misinterpreting vocal pattern since different languages and dialects have different intonation patterns and speech rhythms as earlier stated that a rising intonation at the end of a sentence might be a standard grammatical feature in one language but interpreted as uncertainty or questioning in another.

Q5: Personal Experience Question - What was the most challenging part of this lab for YOU personally, and what was the most surprising or interesting discovery you made?

Your personal experience and discoveries:

The most Challenging part of this lab is handling the ambiguity and nuance in text sentiment. Rule-based systems rely on complex linguistic rules, and machine learning models need to learn subtle patterns from data. Disagreements between VADER and TextBlob, and the lower accuracies of the text-based machine learning models on the small dataset, highlight this complexity which made distinction between the effects of machine learning and rule based model very challenging.

The most Surprising/Interesting is the high accuracy of the audio classifier and the depiction of the results in boxplots showed separation of simulated audio features by emotion as very interesting from data analysis.

🎉 Lab Conclusion

What You've Accomplished

Congratulations! You've successfully:

1. ✅ **Connected all previous modules** to build a comprehensive emotion analysis system
2. ✅ **Built rule-based sentiment analyzers** using VADER and TextBlob
3. ✅ **Created machine learning classifiers** using TF-IDF and scikit-learn
4. ✅ **Developed speech emotion detection** using audio feature extraction
5. ✅ **Implemented multimodal fusion** combining text and audio analysis
6. ✅ **Analyzed bias and ethical considerations** in emotion detection systems

Key Insights from Your Journey

- **Integration is powerful**: Combining concepts from all modules creates more robust systems
- **Different approaches have different strengths**: Rule-based, ML, and multimodal each excel in different scenarios
- **Context matters**: The same words can convey different emotions depending on how they're spoken
- **Bias is real**: Emotion detection systems can perpetuate societal biases and must be carefully evaluated
- **Practical considerations**: Real-world deployment requires balancing accuracy, interpretability, and computational cost

Looking Forward

This lab represents the culmination of your foundational NLP learning. In upcoming modules, you'll learn about:

- **Neural networks** for more sophisticated emotion analysis
- **Deep learning architectures** that can learn complex patterns
- **Transformer models** that understand context even better
- **Large language models** that can perform emotion analysis with minimal training

Final Thought

Remember: The goal isn't just to build accurate models, but to build **fair, ethical, and beneficial systems** that respect human dignity and promote positive outcomes for all users.

Great work completing this comprehensive emotion analysis lab! You now have hands-on experience with the full spectrum of emotion detection approaches, from rule-based methods to multimodal machine learning systems.

🚀 Next Steps and Further Learning

To continue developing your emotion analysis skills:

1. **Explore real datasets**: Try your techniques on actual emotion datasets like IEMOCAP or RAVDESS (You can download RADVES from Kaggle)
2. **Advanced audio features**: Learn about more sophisticated audio features like prosodic contours and voice quality measures
3. **Deep learning approaches**: Explore neural networks for emotion detection (coming in future modules!)
4. **Multimodal architectures**: Study attention mechanisms and more sophisticated fusion techniques
5. **Bias mitigation**: Research techniques for reducing bias in emotion detection systems

📚 Additional Resources:

- **VADER Documentation**: https://github.com/cjhutto/vaderSentiment
- **TextBlob Documentation**: https://textblob.readthedocs.io/
- **Librosa for Audio Analysis**: https://librosa.org/
- **Emotion Recognition Research**: Search for "multimodal emotion recognition" papers
- **Bias in AI**: Research on fairness and bias in machine learning systems

---

Great work completing this comprehensive emotion analysis lab! You now have hands-on experience with multiple approaches to understanding emotions in text and speech. These skills will be valuable as we continue exploring advanced NLP techniques in upcoming

modules.

```
# Create multimodal dataset (text + corresponding audio)
multimodal_texts = {
    'happy': [
        "I'm so excited about this opportunity!",
        "This is absolutely wonderful news!",
        "I love spending time with my family!",
        "What a beautiful day it is today!",
        "I'm thrilled to be here with you all!"
    ],
    'sad': [
        "I'm feeling really down today.",
        "This situation makes me very sad.",
        "I miss my old friends so much.",
        "Everything seems to be going wrong.",
        "I feel so lonely and isolated."
    ],
    'angry': [
        "This is completely unacceptable!",
        "I'm furious about this decision!",
        "How dare you treat me this way!",
        "This makes me so angry and frustrated!",
        "I can't believe this is happening!"
    ],
    'neutral': [
        "The meeting is scheduled for tomorrow.",
        "Please submit your report by Friday.",
        "The weather forecast shows rain.",
        "I need to buy groceries later.",
        "The train arrives at 3:30 PM."
    ]
}

# Create multimodal dataset
multimodal_data = []
print("🎭 Creating Multimodal Dataset (Text + Audio):")

for emotion, texts in multimodal_texts.items():
    print(f"\n🎵 {emotion.capitalize()} examples:")

    for i, text in enumerate(texts):
        # Generate corresponding audio using the function from Part 3
        audio, sr = create_emotional_audio(emotion, duration=2.0)

        # Extract audio features using the function from Part 3
        audio_features = extract_emotional_features(audio, sr)

        # Store the text, the audio features, and the emotion label together
        multimodal_data.append({
            'text': text,
            'audio_features': audio_features,
            'emotion': emotion
        })

        print(f"  Text: {text}")
        if i == 0:  # Play the first audio sample for demonstration
            print(f"  Listen to its corresponding audio:")
            display(Audio(audio, rate=sr))

print(f"\n📊 Multimodal Dataset Created: {len(multimodal_data)} samples")
print("Each sample now contains both text and its corresponding emotional audio features.")
```

```
🎭 Creating Multimodal Dataset (Text + Audio):

🎵 Happy examples:
  Text: I'm so excited about this opportunity!
  Listen to its corresponding audio:

      0:00 / 0:02

  Text: This is absolutely wonderful news!
  Text: I love spending time with my family!
  Text: What a beautiful day it is today!
  Text: I'm thrilled to be here with you all!

🎵 Sad examples:
  Text: I'm feeling really down today.
  Listen to its corresponding audio:

      0:00 / 0:02

  Text: This situation makes me very sad.
  Text: I miss my old friends so much.
  Text: Everything seems to be going wrong.
  Text: I feel so lonely and isolated.

🎵 Angry examples:
  Text: This is completely unacceptable!
  Listen to its corresponding audio:

      0:00 / 0:02

  Text: I'm furious about this decision!
  Text: How dare you treat me this way!
  Text: This makes me so angry and frustrated!
  Text: I can't believe this is happening!

🎵 Neutral examples:
  Text: The meeting is scheduled for tomorrow.
  Listen to its corresponding audio:

      0:00 / 0:02

  Text: Please submit your report by Friday.
  Text: The weather forecast shows rain.
  Text: I need to buy groceries later.
  Text: The train arrives at 3:30 PM.

📊 Multimodal Dataset Created: 20 samples
Each sample now contains both text and its corresponding emotional audio features.
```

```python
# Extract emotional features from audio (building on Module 3)
def extract_emotional_features(audio, sample_rate):
    """Extract features that indicate emotional state"""
    features = {}

    # Prosodic features (how we speak)
    pitches, magnitudes = librosa.piptrack(y=audio, sr=sample_rate)
    pitch_values = []
    for t in range(pitches.shape[1]):
        index = magnitudes[:, t].argmax()
        pitch = pitches[index, t]
        if pitch > 0:
            pitch_values.append(pitch)

    if pitch_values:
        features['pitch_mean'] = np.mean(pitch_values)
        features['pitch_std'] = np.std(pitch_values)
    else:
        features['pitch_mean'] = 0
        features['pitch_std'] = 0

    # Energy features
    rms_energy = librosa.feature.rms(y=audio)[0]
    features['energy_mean'] = np.mean(rms_energy)
    features['energy_std'] = np.std(rms_energy)

    # Spectral features
    spectral_centroids = librosa.feature.spectral_centroid(y=audio, sr=sample_rate)[0]
    features['spectral_centroid_mean'] = np.mean(spectral_centroids)
```