

## ✓ Updated Project Code

# Decoding Emotions through Sentiment Analysis of Social Media Conversation

```
# Step 1: Upload the Dataset
from google.colab import files
uploaded = files.upload()
```

Choose files twitter\_training.csv

- **twitter\_training.csv**(text/csv) - 10325088 bytes, last modified: 03/05/2025 - 100% done  
Saving twitter\_training.csv to twitter\_training.csv

## ✓ Load the Dataset

```
# Step 2: Load the Dataset
import pandas as pd
```

```
# Replace with your actual file name if different
df = pd.read_csv('twitter_training.csv', encoding='latin1')
```

## ✓ Data Exploration

```
df.head()
```

|   | 2401 | Borderlands | Positive | im getting on borderlands and i will murder you all , |
|---|------|-------------|----------|---|
| 0 | 2401 | Borderlands | Positive | I am coming to the borders and I will kill you...     |
| 1 | 2401 | Borderlands | Positive | im getting on borderlands and i will kill you ...     |
| 2 | 2401 | Borderlands | Positive | im coming on borderlands and i will murder you...     |
| 3 | 2401 | Borderlands | Positive | im getting on borderlands 2 and i will murder ...     |
| 4 | 2401 | Borderlands | Positive | im getting into borderlands and i can murder y...     |

Next steps: [Generate code with df](#) [View recommended plots](#) [New interactive sheet](#)

```
print("Shape:", df.shape)
print("Columns:", df.columns.tolist())
df.info()
df.isnull().sum()
df.duplicated().sum()
```

```
Shape: (74681, 4)
Columns: ['2401', 'Borderlands', 'Positive', 'im getting on borderlands and i will murder you all ,']
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74681 entries, 0 to 74680
Data columns (total 4 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   2401                                74681 non-null  int64
1   Borderlands                        74681 non-null  object
2   Positive                          74681 non-null  object
3   im getting on borderlands and i will murder you all ,  73995 non-null  object
dtypes: int64(1), object(3)
memory usage: 2.3+ MB
np.int64(2700)
```

## ✓ Check for Missing Values and Duplicates

```
# Check for missing values
missing_values = df.isnull().sum()
print("Missing Values:\n", missing_values)

# Check for duplicate rows
duplicate_rows = df.duplicated().sum()
print("\nNumber of Duplicate Rows:", duplicate_rows)
```

```

Missing Values:
2401          0
Borderlands   0
Positive      0
im getting on borderlands and i will murder you all , 686
dtype: int64

```

Number of Duplicate Rows: 2700

## Visualize a Few Features

```

# Step 1: Import Libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud

```

```

# Step 2: Load Dataset
df = pd.read_csv("twitter_training.csv", header=None)

```

```

# Confirm number of columns (should be 4)
df.columns = ['ID', 'user', 'emotion', 'tweet']

```

```

# Keep only needed columns
df = df[['emotion', 'tweet']]

```

```

# Remove missing or empty rows
df.dropna(inplace=True)
df = df[df['tweet'].str.strip() != '']

```

```

plt.figure(figsize=(10, 5))
sns.countplot(data=df, x='emotion', order=df['emotion'].value_counts().index, palette='Set2')
plt.title("Distribution of Emotions in Tweets")
plt.xlabel("Emotion")
plt.ylabel("Number of Tweets")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

```

```

<ipython-input-9-53fa63821487>:2: FutureWarning:

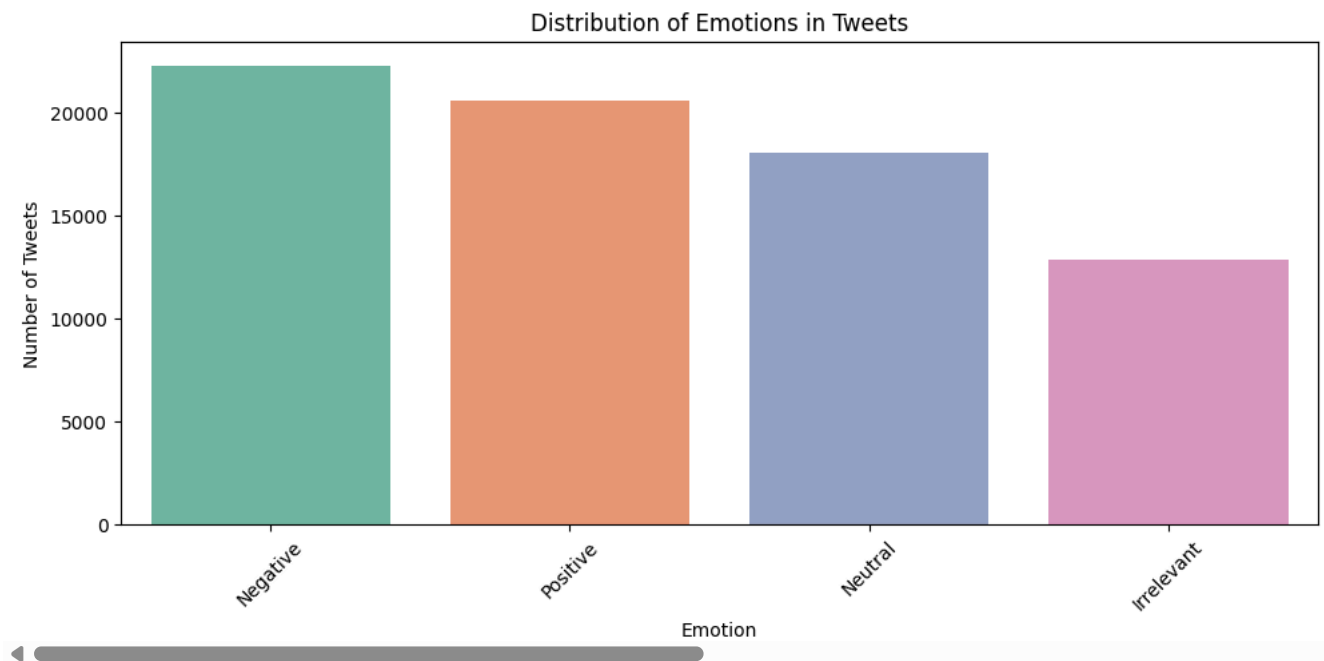
```

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `le`

```

sns.countplot(data=df, x='emotion', order=df['emotion'].value_counts().index, palette='Set2')

```



```

df['tweet_length'] = df['tweet'].apply(lambda x: len(str(x)))

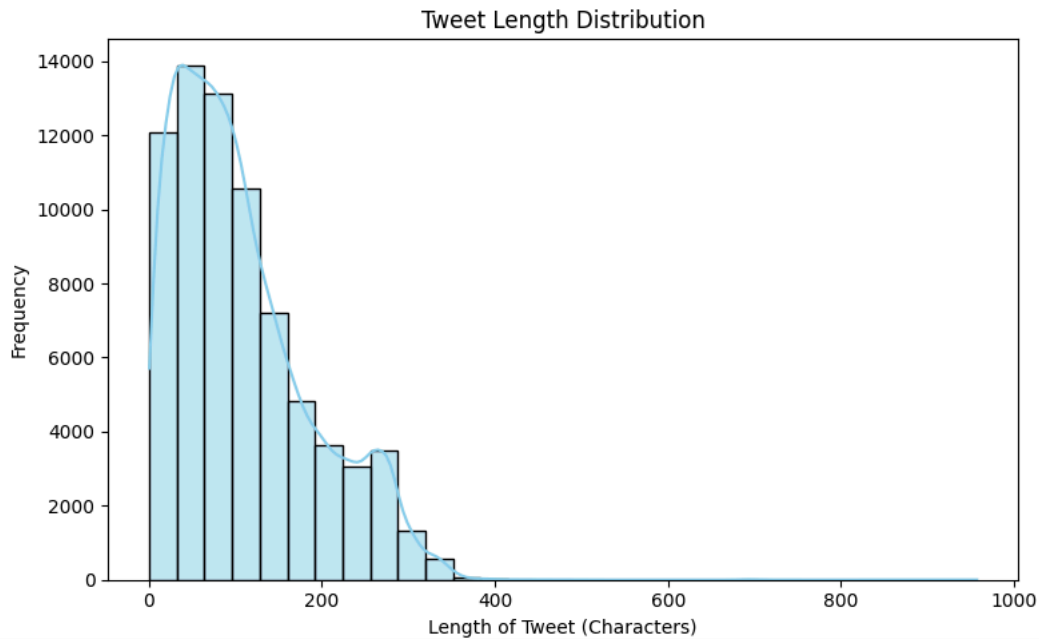
```

```

plt.figure(figsize=(8, 5))
sns.histplot(df['tweet_length'], bins=30, kde=True, color='skyblue')
plt.title("Tweet Length Distribution")
plt.xlabel("Length of Tweet (Characters)")

```

```
plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```



## ✓ Identify Target and Features

```
# Load dataset
import pandas as pd

df = pd.read_csv("twitter_training.csv", header=None)
df.columns = ['ID', 'user', 'emotion', 'tweet']

# Keep only the required columns
df = df[['emotion', 'tweet']]

# Drop missing or empty tweets
df.dropna(inplace=True)
df = df[df['tweet'].str.strip() != '']

# Identify feature and target
X = df['tweet']      # Feature: tweet text
y = df['emotion']    # Target: emotion label

# Display a sample
print(" ♦ Sample Tweet (Feature):\n", X.sample(3).values)
print("\n ♦ Sample Emotion (Target):\n", y.sample(3).values)
```



```
♦ Sample Tweet (Feature):
["Alright I said id go get a clip, played some pc games popped off for the boys that wanted me back. . Getting 3rd partied with no z
Yesterday we played the most intense 1h07m TURBO game in @DOTA2 . It was AMAZING. I had never seen Tier 5 items before hahahah! @t
Assassin's Creed: Narcos.. Sounds pretty cool actually!"]

♦ Sample Emotion (Target):
['Irrelevant' 'Negative' 'Positive']
```

## ✓ Convert Categorical Columns to Numerical

```
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer

# Step 1: Encode the 'emotion' column (target)
le = LabelEncoder()
y_encoded = le.fit_transform(df['emotion'])

# You can map back to emotion names using:
# print(dict(zip(le.classes_, le.transform(le.classes_))))
```

```
# Step 2: Convert 'tweet' text into numeric features using TF-IDF
tfidf = TfidfVectorizer(max_features=5000)
X_tfidf = tfidf.fit_transform(df['tweet']).toarray()
```

```
print("TF-IDF Feature Matrix Shape:", X_tfidf.shape)
print("Encoded Labels Shape:", y_encoded.shape)
```

```
TF-IDF Feature Matrix Shape: (73824, 5000)
Encoded Labels Shape: (73824,)
```

## ✓ One-Hot Encoding

```
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.utils import to_categorical
```

```
# Step 1: Label Encode the target
le = LabelEncoder()
y_encoded = le.fit_transform(df['emotion'])
```

```
# Step 2: One-Hot Encode the label-encoded values
y_onehot = to_categorical(y_encoded)
```

```
# Check the result
print("Shape of One-Hot Encoded Output:", y_onehot.shape)
```

```
Shape of One-Hot Encoded Output: (73824, 4)
```

## ✓ Feature Scaling

```
from sklearn.preprocessing import StandardScaler
```

```
# Apply standard scaling to TF-IDF features
scaler = StandardScaler(with_mean=False) # with_mean=False to avoid error with sparse matrix
X_scaled = scaler.fit_transform(X_tfidf)
```

## ✓ Train-Test Split

```
from sklearn.model_selection import train_test_split
```

```
# Use X_scaled if you've done feature scaling, otherwise use X_tfidf
X = X_scaled # or X_tfidf if not scaled
y = y_encoded # or y_onehot if you're using a deep learning model
```

```
# Split the data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

```
# Check the shape of the result
print("Training set shape:", X_train.shape)
print("Test set shape:", X_test.shape)
```

```
Training set shape: (59059, 5000)
Test set shape: (14765, 5000)
```

## ✓ Model Building

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Initialize the model
model = LogisticRegression(max_iter=1000, random_state=42)
```

```
# Train the model
model.fit(X_train, y_train)
```

```
# Predict on test data
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}\n")

print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=le.classes_))

# Plot confusion matrix
cm = confusion_matrix(y_test, y_pred)

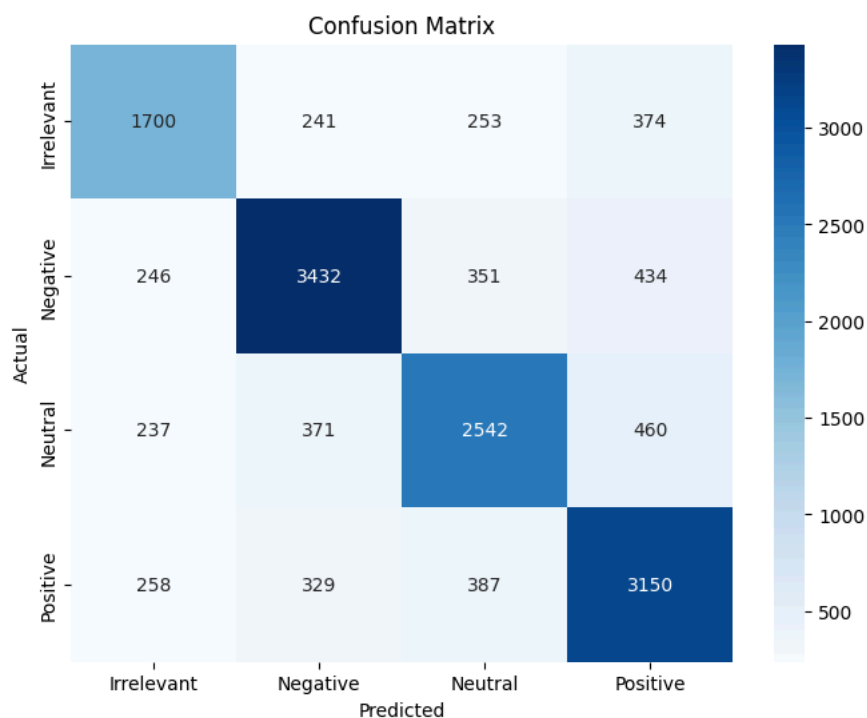
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=le.classes_, yticklabels=le.classes_)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Accuracy: 0.7331

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

 Irrelevant      0.70      0.66      0.68       2568
   Negative      0.78      0.77      0.78      4463
    Neutral      0.72      0.70      0.71      3610
    Positive      0.71      0.76      0.74      4124

 accuracy              0.73
 macro avg              0.72
weighted avg              0.73
```



## ✓ Evaluation

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# 🎯 Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"✅ Accuracy: {accuracy:.4f}")

# 📊 Full Classification Report
print("\n📄 Classification Report:")
print(classification_report(y_test, y_pred, target_names=le.classes_))

# 📊 Confusion Matrix
```

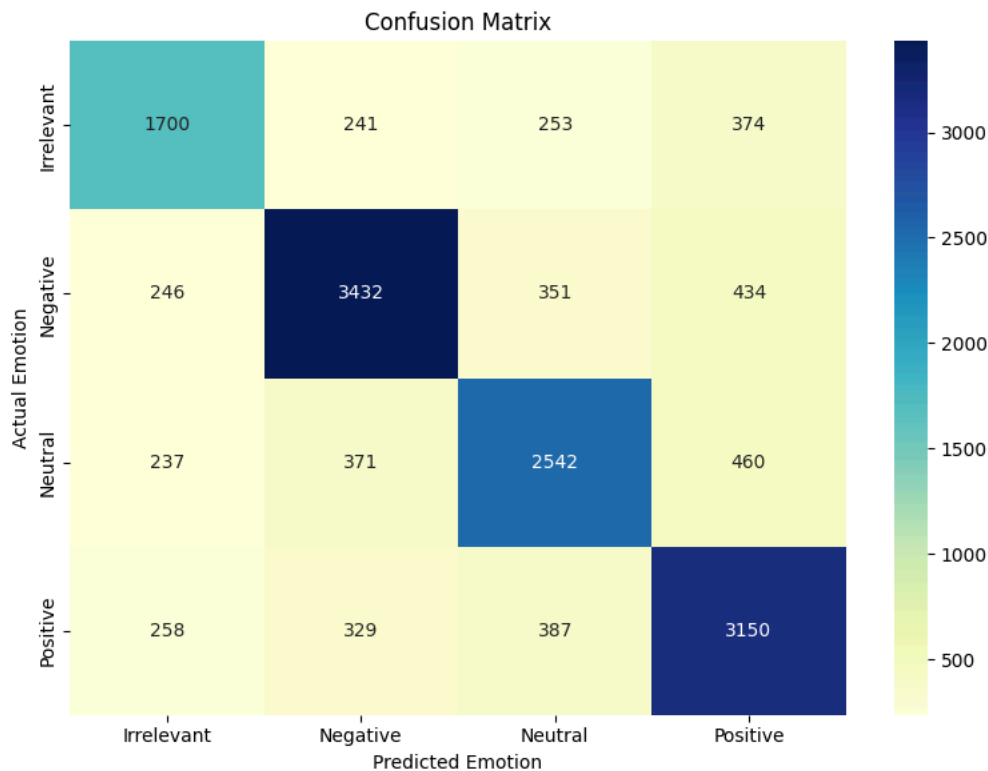
```
cm = confusion_matrix(y_test, y_pred)
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='YlGnBu',
            xticklabels=le.classes_, yticklabels=le.classes_)
plt.title("Confusion Matrix")
plt.xlabel("Predicted Emotion")
plt.ylabel("Actual Emotion")
plt.tight_layout()
plt.show()
```

✓ Accuracy: 0.7331

Classification Report:

|              | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| Irrelevant   | 0.70      | 0.66   | 0.68     | 2568    |
| Negative     | 0.78      | 0.77   | 0.78     | 4463    |
| Neutral      | 0.72      | 0.70   | 0.71     | 3610    |
| Positive     | 0.71      | 0.76   | 0.74     | 4124    |
| accuracy     |           |        | 0.73     | 14765   |
| macro avg    | 0.73      | 0.72   | 0.73     | 14765   |
| weighted avg | 0.73      | 0.73   | 0.73     | 14765   |



## ✓ Make Predictions from New Input

```
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

# Download stopwords if not already done
nltk.download('stopwords')

stop_words = set(stopwords.words('english'))
stemmer = PorterStemmer()

def preprocess(text):
    text = text.lower()
    text = re.sub(r"http\S+|www\S+|https\S+", '', text) # Remove URLs
    text = re.sub(r'@\w+|\#', '', text) # Remove mentions/hashtags
    text = re.sub(r'^a-zA-Z\s', '', text) # Remove punctuation
    words = text.split()
    words = [stemmer.stem(word) for word in words if word not in stop_words]
    return " ".join(words)
```

```

[ntlk_data] Downloading package stopwords to /root/nltk_data...
[ntlk_data] Unzipping corpora/stopwords.zip.

```

```

# Sample new tweet
new_tweet = "I'm so excited and happy about my results!"

# Preprocess the tweet
processed_tweet = preprocess(new_tweet)

# Convert to TF-IDF
new_tfidf = tfidf.transform([processed_tweet])

# Predict
predicted_label = model.predict(new_tfidf)[0]
predicted_emotion = le.inverse_transform([predicted_label])[0]

print(f"🔍 Predicted Emotion: {predicted_emotion}")

```

```

🔍 Predicted Emotion: Positive

```

## ✓ Convert to DataFrame and Encode

```

new_tweet = "I feel terrible and sad about what happened."

import pandas as pd

# Convert new input to a DataFrame
new_df = pd.DataFrame({'tweet': [new_tweet]})

# Apply the same preprocessing function
new_df['clean_tweet'] = new_df['tweet'].apply(preprocess)

# Transform using the previously fitted TF-IDF vectorizer
new_tfidf = tfidf.transform(new_df['clean_tweet'])

# Predict using the trained model
predicted_label = model.predict(new_tfidf)[0]
predicted_emotion = le.inverse_transform([predicted_label])[0]

print(f"🔍 Predicted Emotion: {predicted_emotion}")

```

```

🔍 Predicted Emotion: Negative

```

## ✓ Predict the Final Grade

```

def predict_emotion(tweet):
    # Preprocess
    processed = preprocess(tweet)

    # Encode using TF-IDF
    tfidf_vector = tfidf.transform([processed])

    # Predict with trained model
    label = model.predict(tfidf_vector)[0]

    # Decode to original emotion label
    return le.inverse_transform([label])[0]

sample = "I can't stop smiling today!"
print("Predicted Emotion:", predict_emotion(sample))

```

```

🔍 Predicted Emotion: Positive

```

## ✓ Deployment-Building an Interactive App

```
!pip install gradio
```

```
Collecting semantic-version~=2.0 (from gradio)
  Downloading semantic_version-2.10.0-py2.py3-none-any.whl.metadata (9.7 kB)
Collecting starlette<1.0,>=0.40.0 (from gradio)
  Downloading starlette-0.46.2-py3-none-any.whl.metadata (6.2 kB)
Collecting tomlkit<0.14.0,>=0.12.0 (from gradio)
  Downloading tomlkit-0.13.2-py3-none-any.whl.metadata (2.7 kB)
Requirement already satisfied: typer<1.0,>=0.12 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.15.3)
Requirement already satisfied: typing-extensions~=4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.13.2)
Collecting uvicorn>=0.14.0 (from gradio)
  Downloading uvicorn-0.34.2-py3-none-any.whl.metadata (6.5 kB)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.1->gradio) (2025.3.2)
Requirement already satisfied: websockets<16.0,>=10.0 in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.1->gradio) (13.1)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (3.10)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (1.3.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (2025.4.26)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (1.0.9)
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.11/dist-packages (from httpcore==1.*->httpx>=0.24.1->gradio) (0.14.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (3.18.0)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (2.32.3)
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (4.67.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2.9.0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.2)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (0.7.0)
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (2.33.2)
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio) (0.10.0)
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (8.2.0)
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (1.5.4)
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (13.9.4)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas<3.0,>=1.0) (1.17.0)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0,>=0.12) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0,>=0.12) (2.19.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub) (3.4.0)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub) (2.3.0)
Requirement already satisfied: mdurl~0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich>=10.11.0->typer<1.0,>=0.12) (0.1.2)
Downloading gradio-5.29.1-py3-none-any.whl (54.1 MB)
 54.1/54.1 MB 20.6 MB/s eta 0:00:00
Downloading gradio_client-1.10.1-py3-none-any.whl (323 kB)
 323.1/323.1 kB 22.5 MB/s eta 0:00:00
Downloading aiofiles-24.1.0-py3-none-any.whl (15 kB)
Downloading fastapi-0.115.12-py3-none-any.whl (95 kB)
 95.2/95.2 kB 6.7 MB/s eta 0:00:00
Downloading groovy-0.1.2-py3-none-any.whl (14 kB)
Downloading python_multipart-0.0.20-py3-none-any.whl (24 kB)
Downloading ruff-0.11.10-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.6 MB)
 11.6/11.6 MB 102.5 MB/s eta 0:00:00
Downloading safehttpx-0.1.6-py3-none-any.whl (8.7 kB)
Downloading semantic_version-2.10.0-py2.py3-none-any.whl (15 kB)
Downloading starlette-0.46.2-py3-none-any.whl (72 kB)
 72.0/72.0 kB 6.1 MB/s eta 0:00:00
Downloading tomlkit-0.13.2-py3-none-any.whl (37 kB)
Downloading uvicorn-0.34.2-py3-none-any.whl (62 kB)
 62.5/62.5 kB 4.4 MB/s eta 0:00:00
Downloading ffmpeg-0.5.0-py3-none-any.whl (6.0 kB)
Downloading pydub-0.25.1-py2.py3-none-any.whl (32 kB)
Installing collected packages: pydub, uvicorn, tomlkit, semantic-version, ruff, python-multipart, groovy, ffmpeg, aiofiles, starlette
Successfully installed aiofiles-24.1.0 fastapi-0.115.12 ffmpeg-0.5.0 gradio-5.29.1 gradio-client-1.10.1 groovy-0.1.2 pydub-0.25.1
```

```
import gradio as gr
```

```
# Define a function that does preprocessing + prediction
```

```
def predict_emotion(tweet):
    # Preprocess the tweet
    processed = preprocess(tweet)

    # Convert to TF-IDF
    tfidf_vector = tfidf.transform([processed])

    # Predict the emotion
    label = model.predict(tfidf_vector)[0]
    emotion = le.inverse_transform([label])[0]

    return emotion
```

```
# Define the input and output
```

```
interface = gr.Interface(
    fn=predict_emotion,
    inputs=gr.Textbox(lines=3, placeholder="Type or paste a tweet here..."),
```



```
outputs="text",
title="🔍 Emotion Predictor",
description="Enter a social media post to detect its underlying emotion (e.g., joy, sadness, anger, etc.)"
)

interface.launch()
```

🔗 It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio app to work, sharing must be enabled. Automatica

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()

\* Running on public URL: <https://3c84ac31b6f47efe2e.gradio.live>

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working

## 🔍 Emotion Predictor

Enter a social media post to detect its underlying emotion (e.g., joy, sadness, anger, etc.)

|   |                               |
|---|-------------------------------|
| <div>tweet</div> <div>Type or paste a tweet here...</div> | <div>output</div> <div></div> |
| Clear   | Submit                        |

Flag

Use via API 🚀 · Built with Gradio 🍷 · Settings ⚙️