```
!pip -q install transformers datasets torch accelerate evaluate

import os, re, random, numpy as np, pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt, seaborn as sns
import torch
from datasets import Dataset
from transformers import AutoTokenizer, AutoModel,
AutoModelForSequenceClassification, TrainingArguments, Trainer
import evaluate
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/84.1 kB ? eta -:--:--
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.1/84.1 kB 2.1 MB/s eta
0:00:00
```

```
# CHANGE THIS PATH
FILE_PATH = "/content/drive/MyDrive/NLP PROJECT/Combined Data (1).csv"

df = pd.read_csv(FILE_PATH, encoding="ISO-8859-1",
on_bad_lines="skip", engine="python")

# Basic cleaning
df.drop(columns=[c for c in df.columns if
c.lower().startswith("unnamed")], errors="ignore", inplace=True)
df.dropna(subset=["statement", "status"], inplace=True)
df["statement"] = df["statement"].astype(str)
df["status"] = df["status"].astype(str)

# Keep only valid labels; strip control chars/spaces from status
valid_labels = ["Anxiety", "Bipolar", "Depression", "Normal",
"Personality disorder", "Stress", "Suicidal"]
df["status"] = df["status"].apply(lambda x: re.sub(r"[^\x20-\x7E]",
"", x).strip())
df = df[df["status"].isin(valid_labels)].reset_index(drop=True)

print("□ Cleaned shape:", df.shape)
print("Class counts:\n", df["status"].value_counts())
```

```
□ Cleaned shape: (52517, 2)
Class counts:
 status
Normal                  16213
Depression              15402
Suicidal                10652
Anxiety                  3818
Bipolar                  2775
Stress                   2583
```

```
Personality disorder     1074
Name: count, dtype: int64
```

```
le = LabelEncoder()
df["y"] = le.fit_transform(df["status"])
class_names = list(le.classes_)
num_labels = len(class_names)
print("Classes:", class_names)

X_train_text, X_test_text, y_train, y_test = train_test_split(
    df["statement"].values, df["y"].values,
    test_size=0.2, random_state=42, stratify=df["y"].values
)
print("Train:", len(X_train_text), "Test:", len(X_test_text))
```

```
Classes: ['Anxiety', 'Bipolar', 'Depression', 'Normal', 'Personality
disorder', 'Stress', 'Suicidal']
Train: 42013 Test: 10504
```

```
MODEL_BASE = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(MODEL_BASE)
bert_base = AutoModel.from_pretrained(MODEL_BASE)
device = "cuda" if torch.cuda.is_available() else "cpu"
bert_base.to(device)
bert_base.eval()

@torch.no_grad()
def bert_cls_embeddings(texts, batch_size=64, max_len=128):
    """Return np.array of CLS embeddings (batch-encoded)."""
    vecs = []
    for i in range(0, len(texts), batch_size):
        batch = texts[i:i+batch_size].tolist() if isinstance(texts,
np.ndarray) else texts[i:i+batch_size]
        toks = tokenizer(batch, padding=True, truncation=True,
max_length=max_len, return_tensors="pt")
        toks = {k:v.to(device) for k,v in toks.items()}
        out = bert_base(**toks)                    # last hidden
states (B, L, H)
        cls = out.last_hidden_state[:,0,:]         # CLS token
embedding
        vecs.append(cls.detach().cpu().numpy())
    return np.vstack(vecs)

# Build and cache features (fast ~2–5 min for 53k on GPU)
Xtr_bert = bert_cls_embeddings(X_train_text)
Xte_bert = bert_cls_embeddings(X_test_text)
print("BERT feature shapes:", Xtr_bert.shape, Xte_bert.shape)
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/
_auth.py:94: UserWarning:
```

```
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your
settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to
access public models or datasets.
  warnings.warn(
```

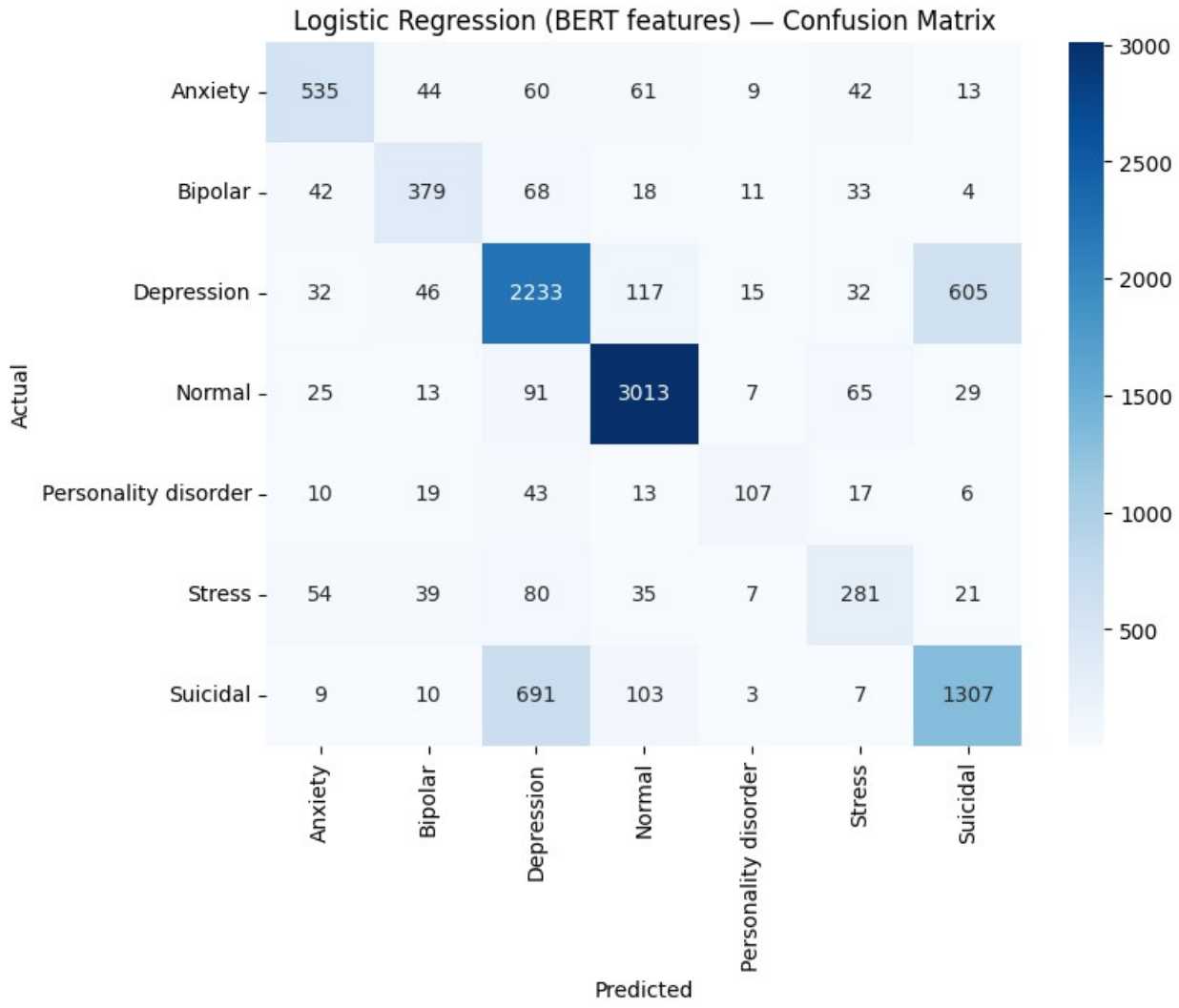

```
{"model_id":"032bf619ebb74e048a22af4cadb20320","version_major":2,"vers
ion_minor":0}

{"model_id":"a12d68e8c0764fa1a472d675f2d45022","version_major":2,"vers
ion_minor":0}

{"model_id":"010454cc370844ed94a43f87feff914e","version_major":2,"vers
ion_minor":0}

{"model_id":"bff0ec21dd5c4e0289ff0ce95509875a","version_major":2,"vers
ion_minor":0}

{"model_id":"ddb0b18fc7bd41cfad4bf3149786de20","version_major":2,"vers
ion_minor":0}
```


```
BERT feature shapes: (42013, 768) (10504, 768)
```

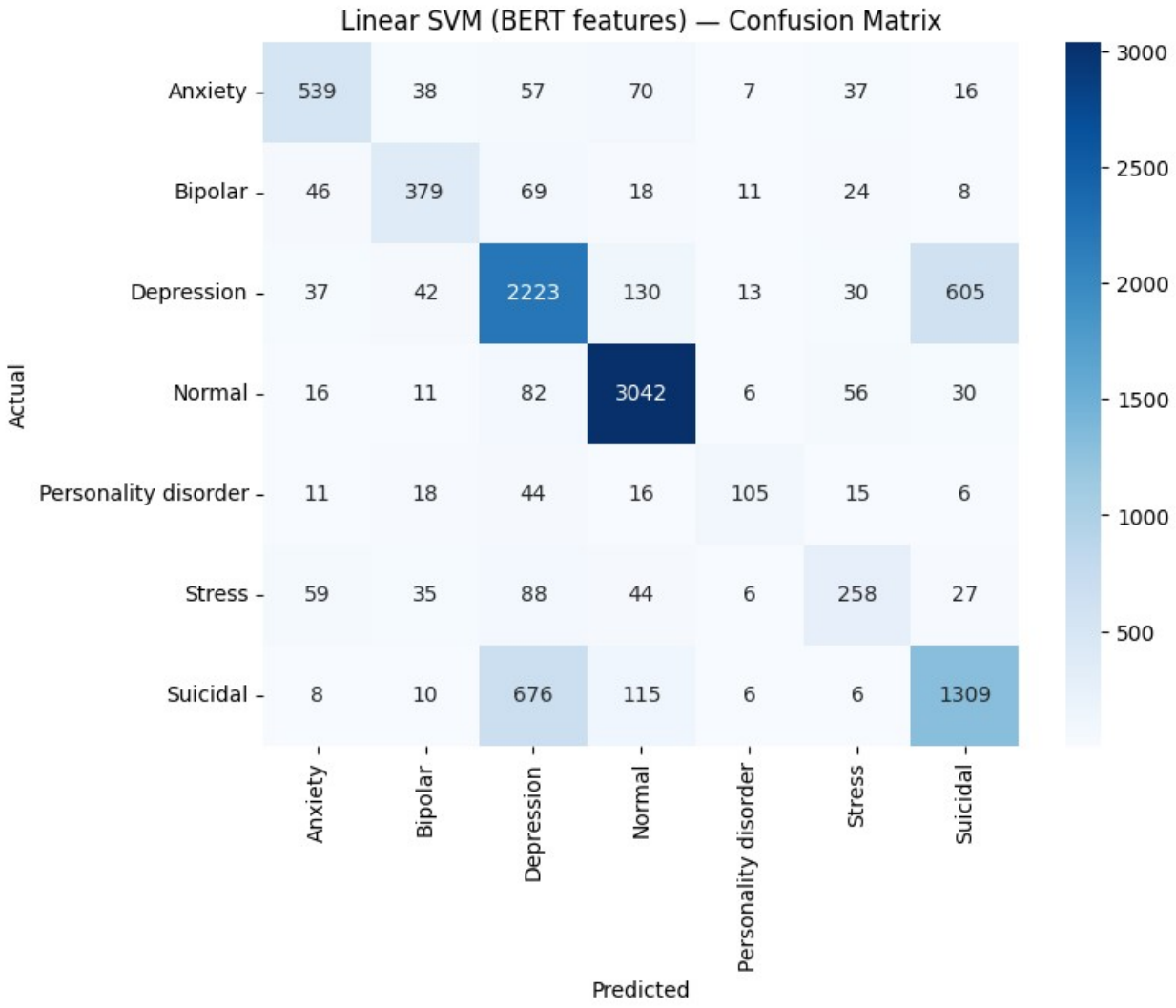
```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier

def eval_model(name, y_true, y_pred, labels):
    print(f"\n=== {name} — Classification Report ===")
    print(classification_report(y_true, y_pred, target_names=labels,
digits=3))
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8,6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=labels, yticklabels=labels)
    plt.title(f"{name} — Confusion Matrix")
    plt.xlabel("Predicted"); plt.ylabel("Actual")
    plt.show()

# Logistic Regression (strong baseline)
lr = LogisticRegression(max_iter=2000, n_jobs=-1)
lr.fit(Xtr_bert, y_train)
pred_lr = lr.predict(Xte_bert)
eval_model("Logistic Regression (BERT features)", y_test, pred_lr,
class_names)

# Linear SVM
svm = LinearSVC()
```

```
svm.fit(Xtr_bert, y_train)
pred_svm = svm.predict(Xte_bert)
eval_model("Linear SVM (BERT features)", y_test, pred_svm,
class_names)

# Random Forest (often weaker on dense embeddings, but included)
rf = RandomForestClassifier(n_estimators=400, max_depth=None, n_jobs=-
1, random_state=42)
rf.fit(Xtr_bert, y_train)
pred_rf = rf.predict(Xte_bert)
eval_model("Random Forest (BERT features)", y_test, pred_rf,
class_names)
```

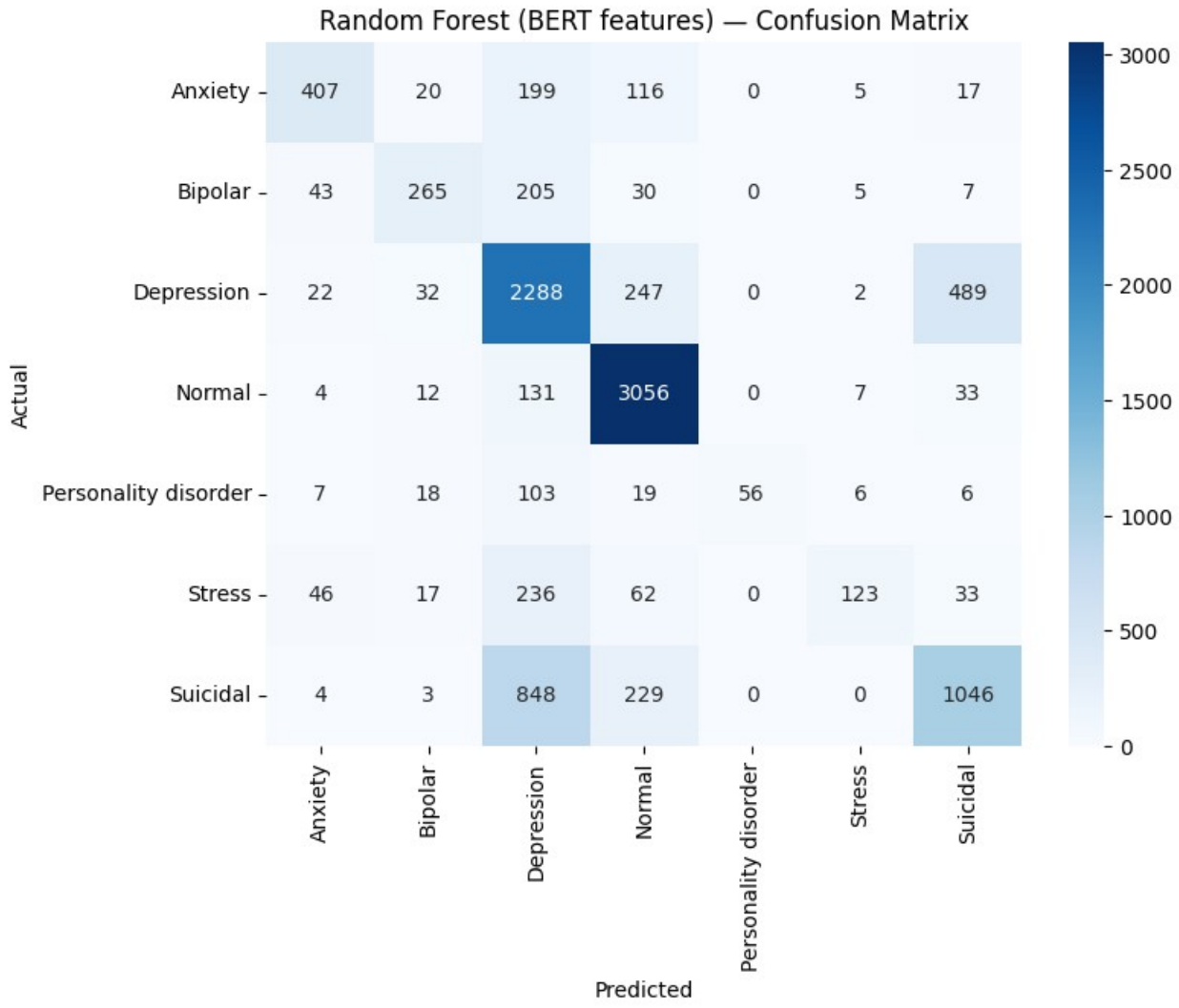
```
=== Logistic Regression (BERT features) — Classification Report ===
                     precision    recall  f1-score   support

            Anxiety      0.757     0.700     0.727       764
            Bipolar      0.689     0.683     0.686       555
         Depression      0.684     0.725     0.704      3080
             Normal      0.897     0.929     0.913      3243
Personality disorder     0.673     0.498     0.572       215
             Stress      0.589     0.544     0.565       517
           Suicidal      0.658     0.614     0.635      2130

           accuracy                          0.748     10504
          macro avg      0.707     0.670     0.686     10504
       weighted avg      0.745     0.748     0.746     10504
```




```
=== Linear SVM (BERT features) — Classification Report ===
                    precision    recall  f1-score   support

           Anxiety      0.753     0.705     0.728       764
           Bipolar      0.711     0.683     0.697       555
        Depression      0.686     0.722     0.704      3080
            Normal      0.886     0.938     0.911      3243
Personality disorder      0.682     0.488     0.569       215
            Stress      0.606     0.499     0.547       517
          Suicidal      0.654     0.615     0.634      2130

          accuracy                          0.748     10504
         macro avg      0.711     0.664     0.684     10504
      weighted avg      0.743     0.748     0.744     10504
```




```
=== Random Forest (BERT features) — Classification Report ===
                    precision    recall  f1-score   support

           Anxiety      0.764     0.533     0.628       764
           Bipolar      0.722     0.477     0.575       555
        Depression      0.571     0.743     0.645      3080
            Normal      0.813     0.942     0.873      3243
Personality disorder    1.000     0.260     0.413       215
            Stress      0.831     0.238     0.370       517
           Suicidal     0.641     0.491     0.556      2130

          accuracy                          0.689     10504
         macro avg      0.763     0.526     0.580     10504
      weighted avg      0.703     0.689     0.674     10504
```




```
!pip install -U transformers
```

```
Requirement already satisfied: transformers in
/usr/local/lib/python3.12/dist-packages (4.57.1)
Requirement already satisfied: filelock in
/usr/local/lib/python3.12/dist-packages (from transformers) (3.20.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.34.0 in
/usr/local/lib/python3.12/dist-packages (from transformers) (0.35.3)
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.12/dist-packages (from transformers) (2.0.2)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.12/dist-packages (from transformers) (25.0)
Requirement already satisfied: pyyaml>=5.1 in
/usr/local/lib/python3.12/dist-packages (from transformers) (6.0.3)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.12/dist-packages (from transformers)
(2024.11.6)
```

```
Requirement already satisfied: requests in
/usr/local/lib/python3.12/dist-packages (from transformers) (2.32.4)
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in
/usr/local/lib/python3.12/dist-packages (from transformers) (0.22.1)
Requirement already satisfied: safetensors>=0.4.3 in
/usr/local/lib/python3.12/dist-packages (from transformers) (0.6.2)
Requirement already satisfied: tqdm>=4.27 in
/usr/local/lib/python3.12/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: fsspec>=2023.5.0 in
/usr/local/lib/python3.12/dist-packages (from huggingface-
hub<1.0,>=0.34.0->transformers) (2025.3.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.12/dist-packages (from huggingface-
hub<1.0,>=0.34.0->transformers) (4.15.0)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in
/usr/local/lib/python3.12/dist-packages (from huggingface-
hub<1.0,>=0.34.0->transformers) (1.1.10)
Requirement already satisfied: charset_normalizer<4,>=2 in
/usr/local/lib/python3.12/dist-packages (from requests->transformers)
(3.4.4)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.12/dist-packages (from requests->transformers)
(3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.12/dist-packages (from requests->transformers)
(2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.12/dist-packages (from requests->transformers)
(2025.10.5)
```

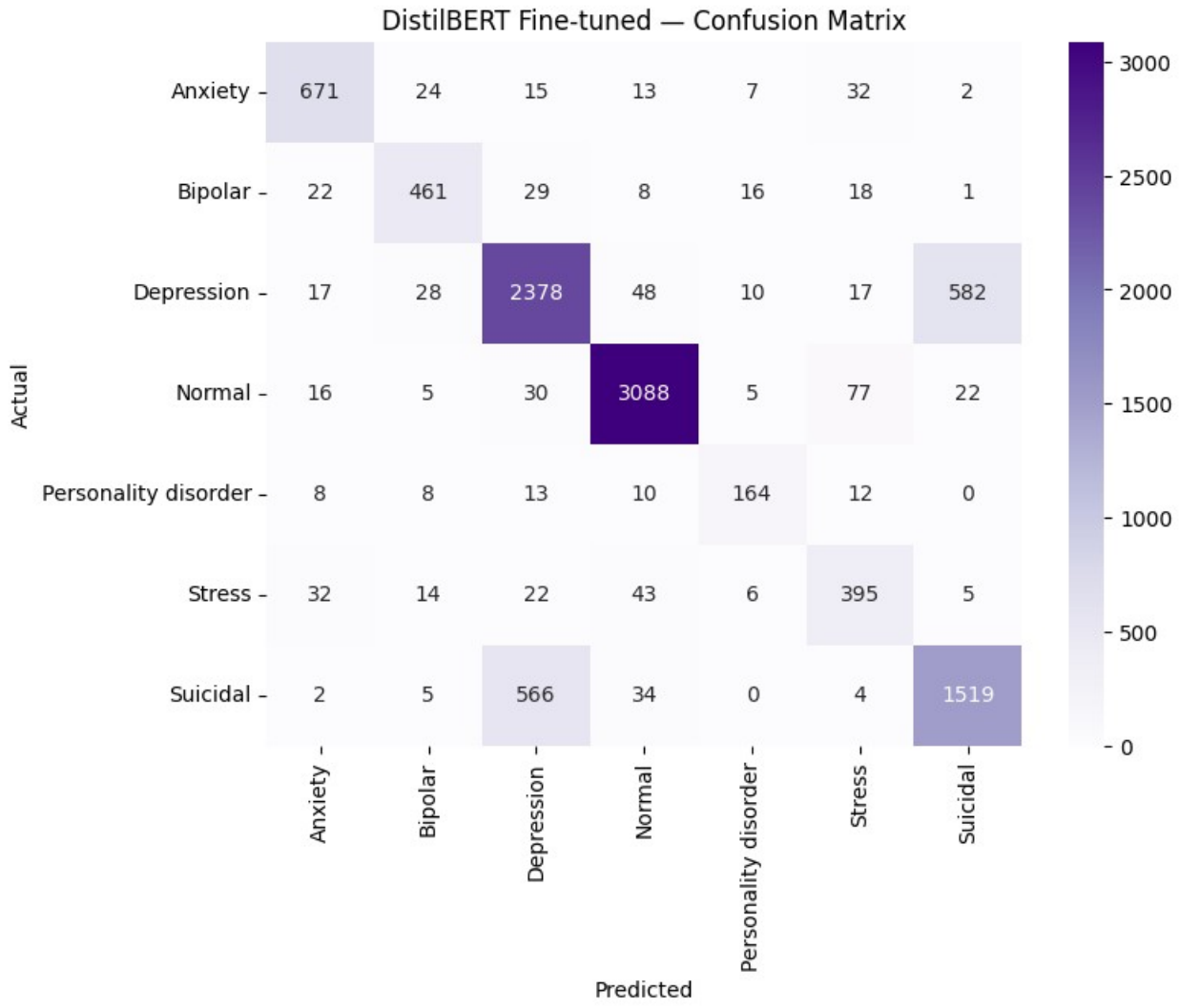
```
# □ DistilBERT fine-tuning — compatible with older transformers (no
evaluation_strategy)

df_train = pd.DataFrame({"text": X_train_text, "labels": y_train})
df_test  = pd.DataFrame({"text": X_test_text, "labels": y_test})

# Datasets
ds_tok_train = Dataset.from_pandas(df_train)
ds_tok_test  = Dataset.from_pandas(df_test)

def tok(batch):
    return tokenizer(batch["text"], truncation=True,
padding="max_length", max_length=128)

ds_tok_train = ds_tok_train.map(tok, batched=True)
ds_tok_test  = ds_tok_test.map(tok, batched=True)

for d in (ds_tok_train, ds_tok_test):
    d.set_format(type="torch", columns=["input_ids", "attention_mask",
"labels"])
```

```
# Model
model_cls =
AutoModelForSequenceClassification.from_pretrained(MODEL_BASE,
num_labels=num_labels).to(device)

# ⬇ Minimal args supported by older versions
args = TrainingArguments(
    output_dir="./bert_ft",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_dir="./logs",
    do_eval=True,          # we'll call evaluate manually after
training
    save_total_limit=1,
    report_to="none"       # disable wandb/tensorboard
)

trainer = Trainer(
    model=model_cls,
    args=args,
    train_dataset=ds_tok_train,
    eval_dataset=ds_tok_test,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics   # already defined earlier
)

print("□ Fine-tuning DistilBERT classifier...")
trainer.train()

# □ Evaluate
out = trainer.predict(ds_tok_test)
y_pred_bert_ft = out.predictions.argmax(-1)

print("\n=== DistilBERT Fine-tuned – Classification Report ===")
print(classification_report(y_test, y_pred_bert_ft,
target_names=class_names, digits=3))

cm = confusion_matrix(y_test, y_pred_bert_ft)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Purples',
            xticklabels=class_names, yticklabels=class_names)
plt.title("DistilBERT Fine-tuned – Confusion Matrix")
plt.xlabel("Predicted"); plt.ylabel("Actual")
plt.show()
```

{"model_id":"a2b7a29712404e24a8213238f307b958","version_major":2,"version_minor":0}

{"model_id":"2f07ada107b14e828e2ca6bcae1b4769","version_major":2,"version_minor":0}

```
Some weights of DistilBertForSequenceClassification were not
initialized from the model checkpoint at distilbert-base-uncased and
are newly initialized: ['classifier.bias', 'classifier.weight',
'pre_classifier.bias', 'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
/tmp/ipython-input-3793599348.py:36: FutureWarning: `tokenizer` is
deprecated and will be removed in version 5.0.0 for
`Trainer.__init__`. Use `processing_class` instead.
  trainer = Trainer(

 Fine-tuning DistilBERT classifier...

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>


=== DistilBERT Fine-tuned — Classification Report ===
                      precision    recall  f1-score   support

             Anxiety      0.874     0.878     0.876       764
             Bipolar      0.846     0.831     0.838       555
          Depression      0.779     0.772     0.775      3080
              Normal      0.952     0.952     0.952      3243
Personality disorder      0.788     0.763     0.775       215
              Stress      0.712     0.764     0.737       517
            Suicidal      0.713     0.713     0.713      2130

            accuracy                          0.826     10504
           macro avg      0.809     0.810     0.810     10504
        weighted avg      0.826     0.826     0.826     10504
```




```python
@torch.no_grad()
def bert_cls_embeddings(texts, batch_size=64, max_len=128):
    """Return [CLS] embeddings for given list of texts."""
    # Ensure input is a plain Python list of strings
    if isinstance(texts, (np.ndarray, pd.Series)):
        texts = texts.tolist()
    vecs = []
    for i in range(0, len(texts), batch_size):
        batch = texts[i:i+batch_size]
        toks = tokenizer(batch, padding=True, truncation=True,
max_length=max_len, return_tensors="pt").to(device)
        out = bert_base(**toks)
        cls = out.last_hidden_state[:, 0, :]  # CLS token embedding
        vecs.append(cls.cpu().numpy())
    return np.vstack(vecs)
```

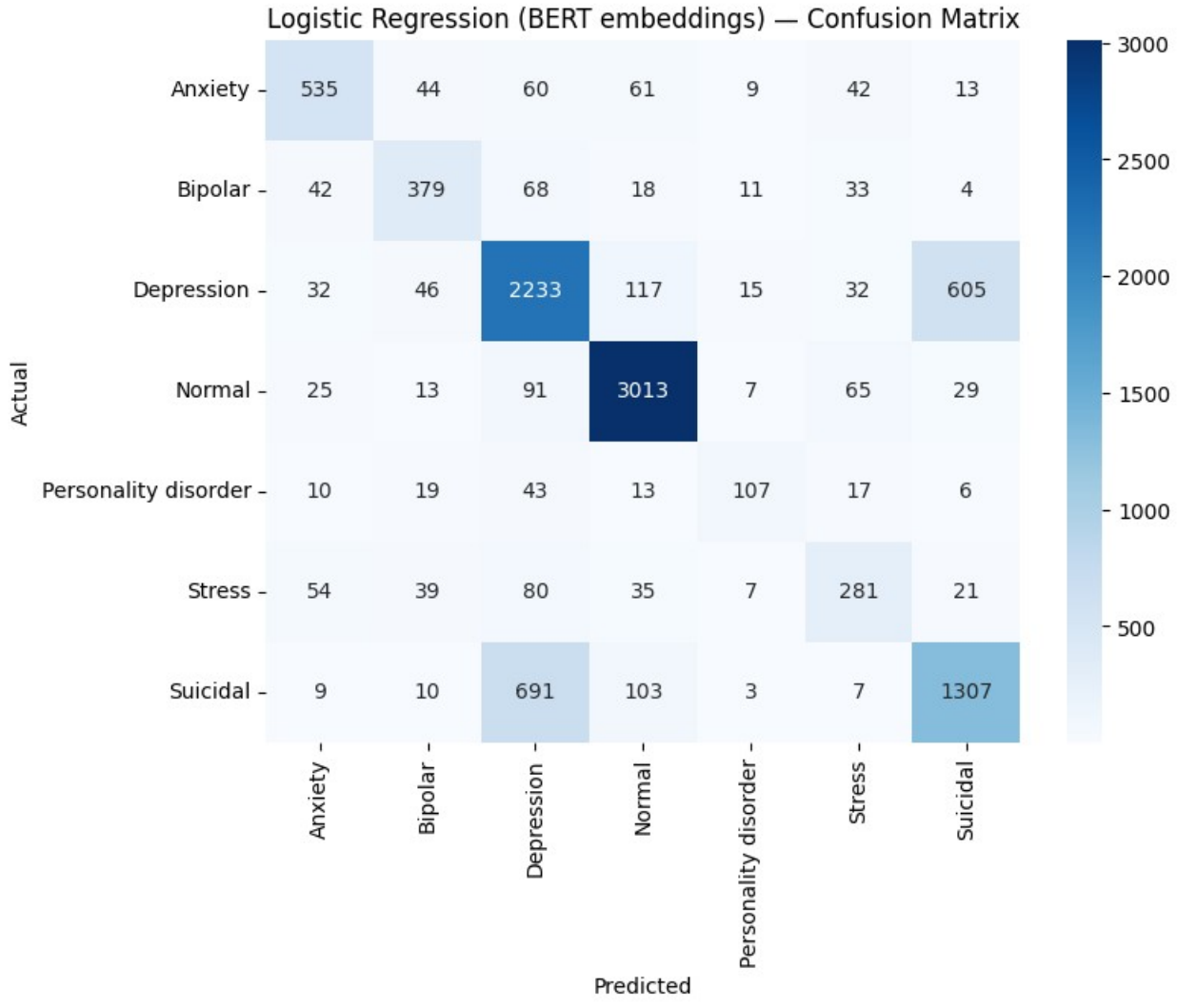
```
print("⚙ Generating BERT embeddings (≈5 min on GPU)...")
Xtr_bert = bert_cls_embeddings(X_train_text)
Xte_bert = bert_cls_embeddings(X_test_text)
print("✅ Done. Shapes:", Xtr_bert.shape, Xte_bert.shape)
```

```
⚙ Generating BERT embeddings (≈5 min on GPU)...
✅ Done. Shapes: (42013, 768) (10504, 768)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns, matplotlib.pyplot as plt

def eval_model(name, y_true, y_pred, labels):
    print(f"\n=== {name} — Classification Report ===")
    print(classification_report(y_true, y_pred, target_names=labels,
digits=3))
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8,6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=labels, yticklabels=labels)
    plt.title(f"{name} — Confusion Matrix")
    plt.xlabel("Predicted"); plt.ylabel("Actual")
    plt.show()

# 🔹 Logistic Regression
lr = LogisticRegression(max_iter=2000, n_jobs=-1)
lr.fit(Xtr_bert, y_train)
pred_lr = lr.predict(Xte_bert)
eval_model("Logistic Regression (BERT embeddings)", y_test, pred_lr,
class_names)

# 🔹 Linear SVM
svm = LinearSVC()
svm.fit(Xtr_bert, y_train)
pred_svm = svm.predict(Xte_bert)
eval_model("Linear SVM (BERT embeddings)", y_test, pred_svm,
class_names)

# 🔹 Random Forest
rf = RandomForestClassifier(n_estimators=300, random_state=42,
n_jobs=-1)
rf.fit(Xtr_bert, y_train)
pred_rf = rf.predict(Xte_bert)
eval_model("Random Forest (BERT embeddings)", y_test, pred_rf,
class_names)
```

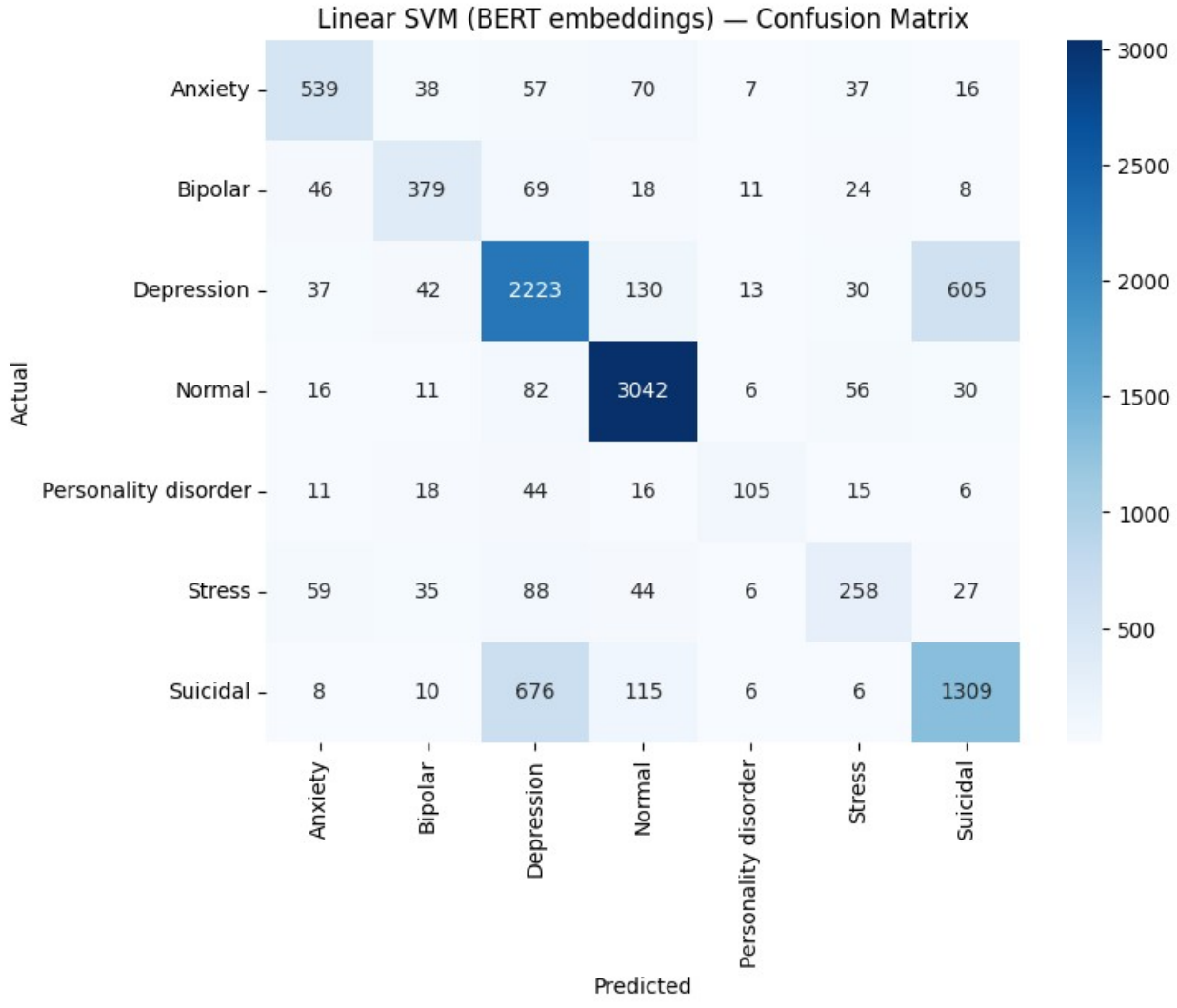
```
=== Logistic Regression (BERT embeddings) — Classification Report ===
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Anxiety | 0.757 | 0.700 | 0.727 | 764 |
| Bipolar | 0.689 | 0.683 | 0.686 | 555 |
| Depression | 0.684 | 0.725 | 0.704 | 3080 |
| Normal | 0.897 | 0.929 | 0.913 | 3243 |
| Personality disorder | 0.673 | 0.498 | 0.572 | 215 |
| Stress | 0.589 | 0.544 | 0.565 | 517 |
| Suicidal | 0.658 | 0.614 | 0.635 | 2130 |
| accuracy | | | 0.748 | 10504 |
| macro avg | 0.707 | 0.670 | 0.686 | 10504 |
| weighted avg | 0.745 | 0.748 | 0.746 | 10504 |



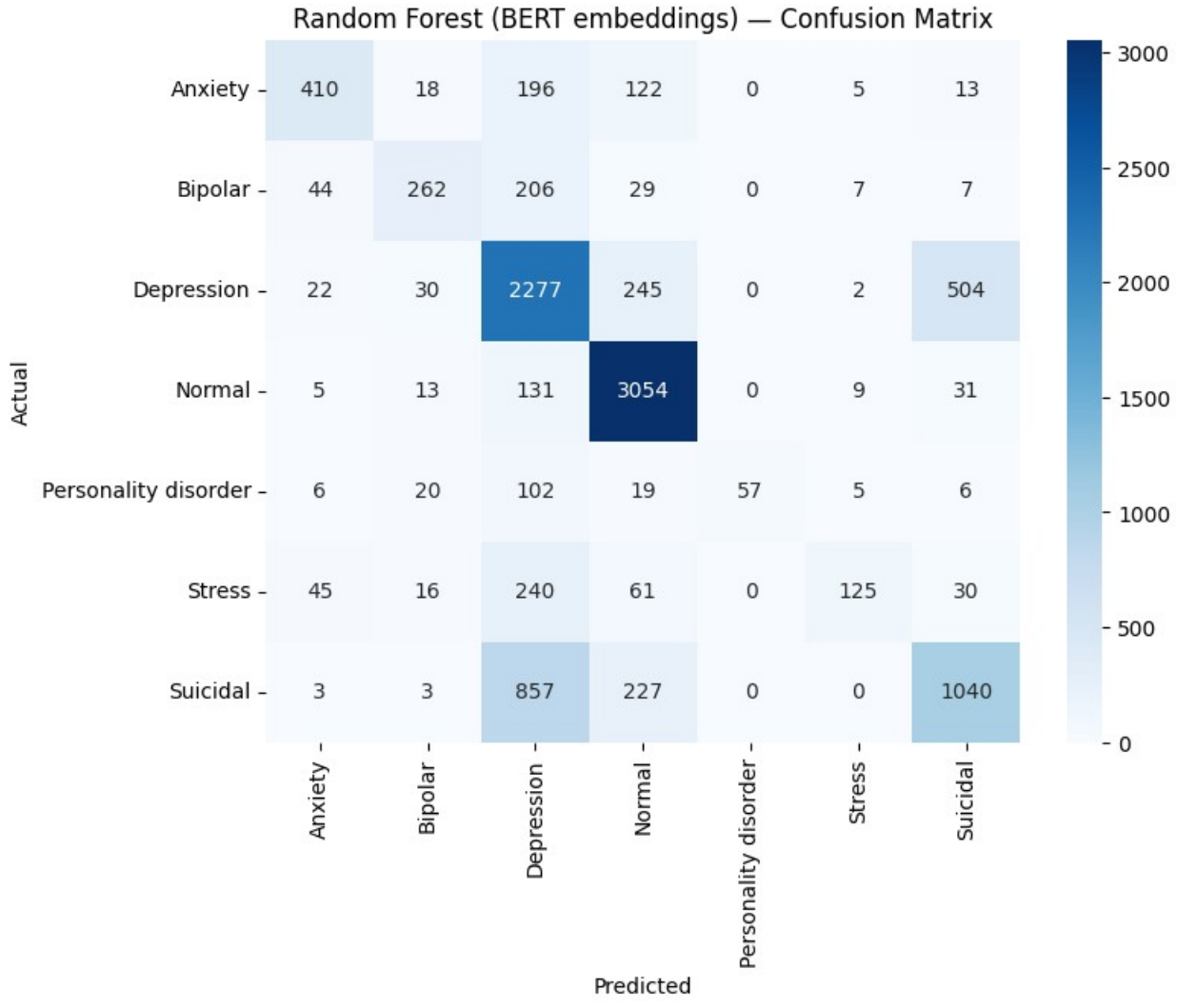

=== Linear SVM (BERT embeddings) — Classification Report ===

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Anxiety | 0.753 | 0.705 | 0.728 | 764 |
| Bipolar | 0.711 | 0.683 | 0.697 | 555 |
| Depression | 0.686 | 0.722 | 0.704 | 3080 |
| Normal | 0.886 | 0.938 | 0.911 | 3243 |
| Personality disorder | 0.682 | 0.488 | 0.569 | 215 |
| Stress | 0.606 | 0.499 | 0.547 | 517 |
| Suicidal | 0.654 | 0.615 | 0.634 | 2130 |
| accuracy | | | 0.748 | 10504 |
| macro avg | 0.711 | 0.664 | 0.684 | 10504 |
| weighted avg | 0.743 | 0.748 | 0.744 | 10504 |




=== Random Forest (BERT embeddings) — Classification Report ===

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Anxiety | 0.766 | 0.537 | 0.631 | 764 |
| Bipolar | 0.724 | 0.472 | 0.571 | 555 |
| Depression | 0.568 | 0.739 | 0.642 | 3080 |
| Normal | 0.813 | 0.942 | 0.873 | 3243 |
| Personality disorder | 1.000 | 0.265 | 0.419 | 215 |
| Stress | 0.817 | 0.242 | 0.373 | 517 |
| Suicidal | 0.638 | 0.488 | 0.553 | 2130 |
| accuracy | | | 0.688 | 10504 |
| macro avg | 0.761 | 0.526 | 0.580 | 10504 |
| weighted avg | 0.701 | 0.688 | 0.673 | 10504 |




```
from sklearn.metrics import accuracy_score, f1_score
import pandas as pd
```

```
results = pd.DataFrame({
    "Model": ["Logistic Regression", "Linear SVM", "Random Forest"],
    "Accuracy": [
        accuracy_score(y_test, pred_lr),
        accuracy_score(y_test, pred_svm),
        accuracy_score(y_test, pred_rf)
    ],
    "Macro F1": [
        f1_score(y_test, pred_lr, average="macro"),
        f1_score(y_test, pred_svm, average="macro"),
        f1_score(y_test, pred_rf, average="macro")
    ]
})

print("📊 Summary — Classical Models on BERT embeddings")
display(results.round(3))
```

```
📊 Summary — Classical Models on BERT embeddings
```


```
{"summary":"{\n  \"name\": \"display(results\",\n  \"rows\": 3,\n
\"fields\": [\n    {\n      \"column\": \"Model\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n
\"Logistic Regression\",\n          \"Linear SVM\",\n
\"Random Forest\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Accuracy\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.03464101615137758,\n        \"min\":
0.688,\n        \"max\": 0.748,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          0.688,\n          0.748\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Macro F1\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.06063002556489653,\n        \"min\": 0.58,\n        \"max\": 0.686,\
n        \"num_unique_values\": 3,\n        \"samples\": [\n
0.686,\n          0.684\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe"}
```


```
def predict_new(texts, clf):
    embs = bert_cls_embeddings(texts)
    preds = clf.predict(embs)
    return le.inverse_transform(preds)

new_texts = [
    "I can’t sleep and my thoughts are racing all night.",
# Anxiety
    "Yesterday I cleaned the entire house in excitement, today I can’t
move.", # Bipolar
```

```
    "Everything feels dull and meaningless lately.",
# Depression
    "Too many deadlines are making me anxious and tired.",
# Stress
    "I feel perfectly fine today, calm and happy.",
# Normal
    "I don’t want to live anymore, I’m tired of everything.",
# Suicidal
    "I can’t control my emotions and my relationships always fall
apart." # Personality disorder
]

for model_name, clf in [("Logistic Regression", lr), ("Linear SVM",
svm), ("Random Forest", rf)]:
    preds = predict_new(new_texts, clf)
    print(f"\n {model_name} Predictions:")
    display(pd.DataFrame({"Statement": new_texts, "Predicted Label":
preds}))
```

```
 Logistic Regression Predictions:
```


```
{"summary":"{\n  \"name\": \"    display(pd\",\n  \"rows\": 7,\n
\"fields\": [\n    {\n      \"column\": \"Statement\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 7,\n        \"samples\": [\n          \"I can\\
u2019t sleep and my thoughts are racing all night.\",\n
\"Yesterday I cleaned the entire house in excitement, today I can\\
u2019t move.\",\n          \"I don\\u2019t want to live anymore, I\\
u2019m tired of everything.\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Predicted Label\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"Normal\",\n          \"Anxiety\",\n
\"Depression\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
 Linear SVM Predictions:
```


```
{"summary":"{\n  \"name\": \"    display(pd\",\n  \"rows\": 7,\n
\"fields\": [\n    {\n      \"column\": \"Statement\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 7,\n        \"samples\": [\n          \"I can\\
u2019t sleep and my thoughts are racing all night.\",\n
\"Yesterday I cleaned the entire house in excitement, today I can\\
u2019t move.\",\n          \"I don\\u2019t want to live anymore, I\\
u2019m tired of everything.\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Predicted Label\",\n      \"properties\": {\n
```

\"dtype\": \"category\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"Anxiety\",\n          \"Normal\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

 Random Forest Predictions:

{"summary":"{\n  \"name\": \"    display(pd\",\n  \"rows\": 7,\n
\"fields\": [\n    {\n      \"column\": \"Statement\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 7,\n        \"samples\": [\n          \"I can\\
u2019t sleep and my thoughts are racing all night.\",\n
\"Yesterday I cleaned the entire house in excitement, today I can\\
u2019t move.\",\n          \"I don\\u2019t want to live anymore, I\\
u2019m tired of everything.\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Predicted Label\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"Depression\",\n          \"Normal\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC

# Re-train with class weighting
lr_bal = LogisticRegression(max_iter=2000, class_weight="balanced",
n_jobs=-1)
lr_bal.fit(Xtr_bert, y_train)
pred_lr_bal = lr_bal.predict(Xte_bert)

svm_bal = LinearSVC(class_weight="balanced")
svm_bal.fit(Xtr_bert, y_train)
pred_svm_bal = svm_bal.predict(Xte_bert)

# Evaluate again
print("\n=== Logistic Regression (Balanced) ===")
print(classification_report(y_test, pred_lr_bal,
target_names=class_names, digits=3))

print("\n=== Linear SVM (Balanced) ===")
print(classification_report(y_test, pred_svm_bal,
target_names=class_names, digits=3))
```

```
=== Logistic Regression (Balanced) ===
                    precision    recall  f1-score   support

           Anxiety      0.666     0.730     0.697       764
           Bipolar      0.583     0.750     0.656       555
```

```
          Depression      0.774     0.578     0.662      3080
              Normal      0.926     0.884     0.905      3243
Personality disorder      0.336     0.707     0.455       215
              Stress      0.457     0.665     0.542       517
            Suicidal      0.634     0.701     0.666      2130

            accuracy                          0.724     10504
           macro avg      0.625     0.716     0.655     10504
        weighted avg      0.750     0.724     0.730     10504


=== Linear SVM (Balanced) ===
                        precision    recall  f1-score   support

             Anxiety      0.693     0.742     0.717       764
             Bipolar      0.606     0.744     0.668       555
          Depression      0.761     0.617     0.681      3080
              Normal      0.916     0.910     0.913      3243
Personality disorder      0.337     0.656     0.445       215
              Stress      0.486     0.634     0.550       517
            Suicidal      0.644     0.663     0.653      2130

            accuracy                          0.734     10504
           macro avg      0.635     0.709     0.661     10504
        weighted avg      0.750     0.734     0.738     10504


from sklearn.preprocessing import normalize
probs = lr_bal.predict_proba(Xte_bert)
probs = normalize(probs, norm='l1', axis=1)  # re-scale
pred_lr_thresh = np.argmax(probs, axis=1)

for model_name, clf in [("Logistic Regression (balanced)", lr_bal),
("Linear SVM (balanced)", svm_bal)]:
    preds = predict_new(new_texts, clf)
    print(f"\n {model_name} Predictions:")
    display(pd.DataFrame({"Statement": new_texts, "Predicted Label":
preds}))


 Logistic Regression (balanced) Predictions:

{"summary":"{\n  \"name\": \"    display(pd\",\n  \"rows\": 7,\n
\"fields\": [\n    {\n      \"column\": \"Statement\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 7,\n        \"samples\": [\n          \"I can\\
u2019t sleep and my thoughts are racing all night.\",\n
\"Yesterday I cleaned the entire house in excitement, today I can\\
u2019t move.\",\n          \"I don\\u2019t want to live anymore, I\\
u2019m tired of everything.\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
```

\"column\": \"Predicted Label\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"Stress\",\n          \"Normal\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

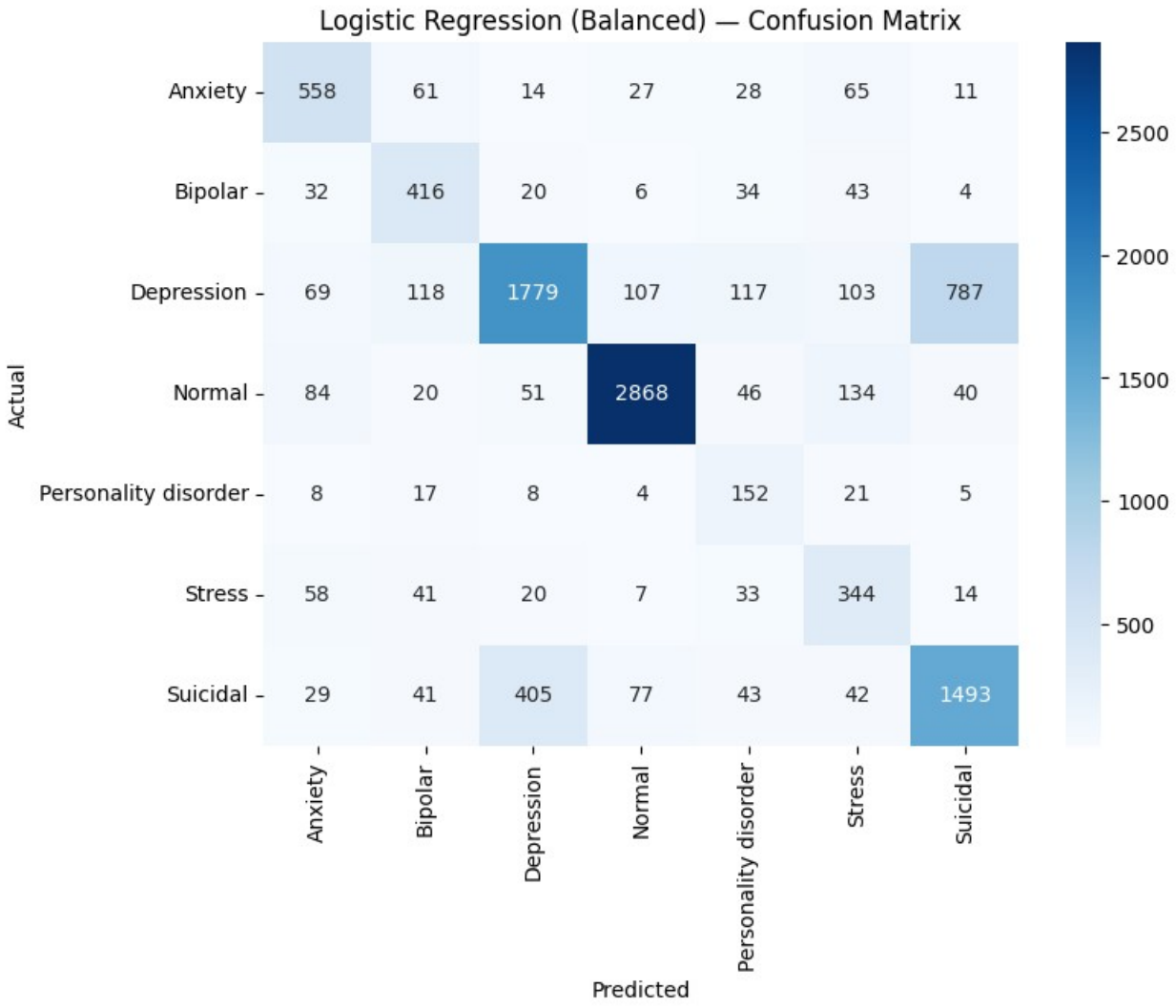
 Linear SVM (balanced) Predictions:

{"summary":"{\n  \"name\": \"    display(pd\",\n  \"rows\": 7,\n
\"fields\": [\n    {\n      \"column\": \"Statement\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 7,\n        \"samples\": [\n          \"I can\\
u2019t sleep and my thoughts are racing all night.\",\n
\"Yesterday I cleaned the entire house in excitement, today I can\\
u2019t move.\",\n          \"I don\\u2019t want to live anymore, I\\
u2019m tired of everything.\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Predicted Label\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"Normal\",\n          \"Anxiety\",\n
\"Suicidal\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

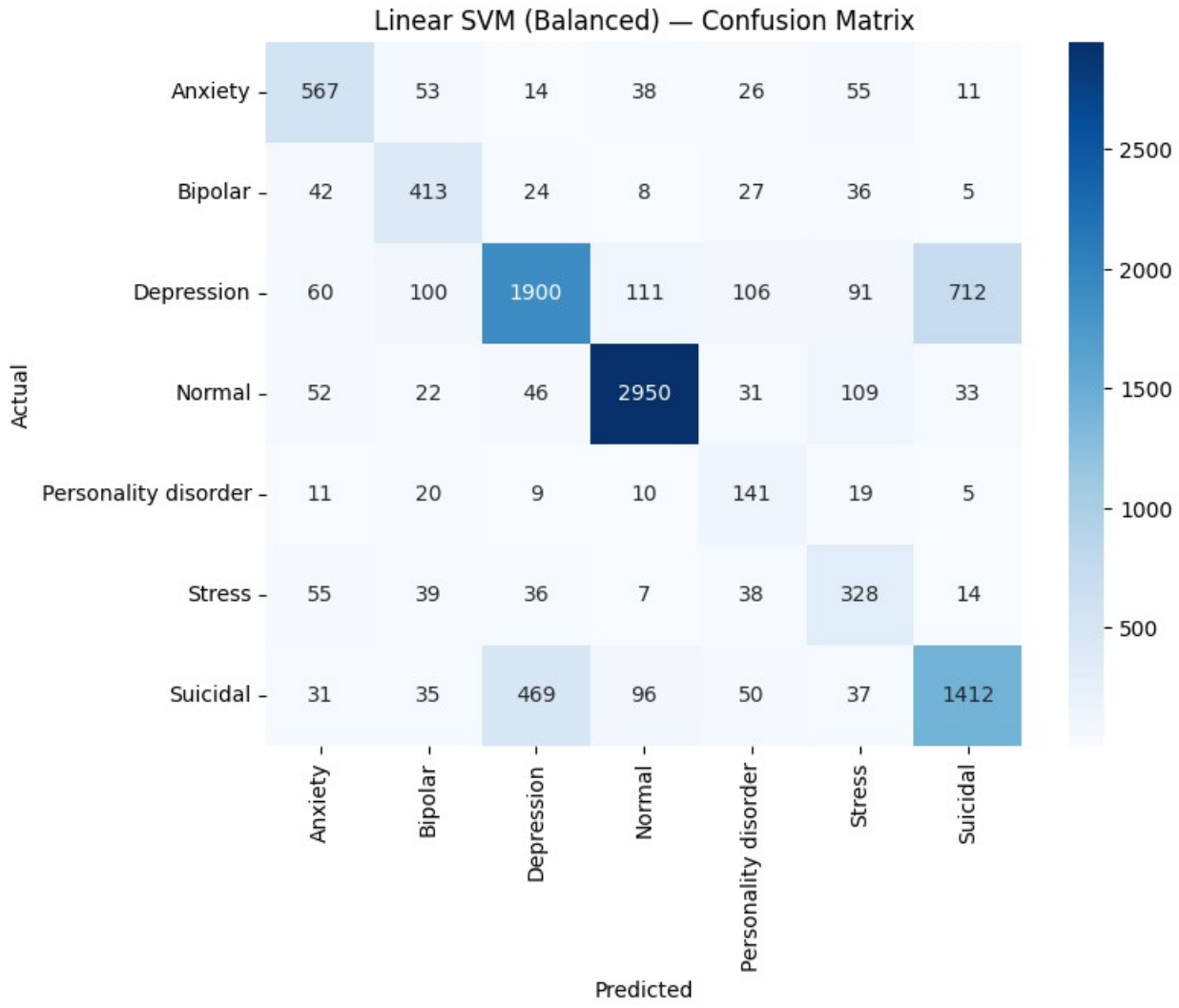
```
extra = {
    "Bipolar": [
        "I felt unstoppable yesterday, today I can’t even get out of
bed.",
        "My energy swings from sky high to drained in hours."
    ],
    "Personality disorder": [
        "I change my opinions about people in seconds.",
        "I can’t hold stable relationships; my emotions flip
instantly."
    ],
    "Stress": [
        "I can’t focus; my deadlines keep piling up.",
        "Every small task feels overwhelming lately."
    ]
}

for label, sentences in extra.items():
    for s in sentences:
        df = pd.concat([df, pd.DataFrame({"statement":[s], "status":
[label]})], ignore_index=True)

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

```
def eval_model(name, y_true, y_pred, labels):
    print(f"\n=== {name} — Classification Report ===")
    print(classification_report(y_true, y_pred, target_names=labels,
digits=3))
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8,6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=labels, yticklabels=labels)
    plt.title(f"{name} — Confusion Matrix")
    plt.xlabel("Predicted"); plt.ylabel("Actual")
    plt.show()

# ⚖️ Logistic Regression with class weighting
lr_bal = LogisticRegression(max_iter=2000, class_weight="balanced",
n_jobs=-1)
lr_bal.fit(Xtr_bert, y_train)
pred_lr_bal = lr_bal.predict(Xte_bert)
eval_model("Logistic Regression (Balanced)", y_test, pred_lr_bal,
class_names)

# ⚖️ Linear SVM with class weighting
svm_bal = LinearSVC(class_weight="balanced")
svm_bal.fit(Xtr_bert, y_train)
pred_svm_bal = svm_bal.predict(Xte_bert)
eval_model("Linear SVM (Balanced)", y_test, pred_svm_bal, class_names)
```

```
=== Logistic Regression (Balanced) — Classification Report ===
                     precision    recall  f1-score   support

            Anxiety      0.666     0.730     0.697       764
            Bipolar      0.583     0.750     0.656       555
         Depression      0.774     0.578     0.662      3080
             Normal      0.926     0.884     0.905      3243
Personality disorder      0.336     0.707     0.455       215
             Stress      0.457     0.665     0.542       517
           Suicidal      0.634     0.701     0.666      2130

           accuracy                          0.724     10504
          macro avg      0.625     0.716     0.655     10504
       weighted avg      0.750     0.724     0.730     10504
```




```
=== Linear SVM (Balanced) — Classification Report ===
                    precision    recall  f1-score   support

           Anxiety      0.693     0.742     0.717       764
           Bipolar      0.606     0.744     0.668       555
        Depression      0.761     0.617     0.681      3080
            Normal      0.916     0.910     0.913      3243
Personality disorder    0.337     0.656     0.445       215
            Stress      0.486     0.634     0.550       517
           Suicidal     0.644     0.663     0.653      2130

          accuracy                          0.734     10504
         macro avg      0.635     0.709     0.661     10504
      weighted avg      0.750     0.734     0.738     10504
```




```
from sklearn.metrics import accuracy_score, f1_score
import pandas as pd

results_bal = pd.DataFrame({
    "Model": ["Logistic Regression (Balanced)", "Linear SVM
(Balanced)"],
    "Accuracy": [
        accuracy_score(y_test, pred_lr_bal),
        accuracy_score(y_test, pred_svm_bal)
    ],
    "Macro F1": [
        f1_score(y_test, pred_lr_bal, average="macro"),
        f1_score(y_test, pred_svm_bal, average="macro")
    ]
})
```

```
print("□ Summary — Balanced Models")
display(results_bal.round(3))
```

```
□ Summary — Balanced Models
```


```
{"summary":"{\n  \"name\": \"display(results_bal\",\n  \"rows\": 2,\n
\"fields\": [\n    {\n      \"column\": \"Model\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 2,\n        \"samples\": [\n          \"Linear
SVM (Balanced)\",\n          \"Logistic Regression (Balanced)\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Accuracy\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.007071067811865481,\n        \"min\": 0.724,\n        \"max\":
0.734,\n        \"num_unique_values\": 2,\n        \"samples\": [\n
0.734,\n          0.724\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Macro F1\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.004242640687119289,\n        \"min\":
0.655,\n        \"max\": 0.661,\n        \"num_unique_values\": 2,\n
\"samples\": [\n          0.661,\n          0.655\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


```
def predict_new_bal(texts, clf):
    embs = bert_cls_embeddings(texts)   # reuse your existing
embedding extractor
    preds = clf.predict(embs)
    return le.inverse_transform(preds)

new_texts = [
    "I can’t sleep and my thoughts are racing all night.",
# Anxiety
    "Yesterday I cleaned the entire house in excitement, today I can’t
move.", # Bipolar
    "Everything feels dull and meaningless lately.",
# Depression
    "Too many deadlines are making me anxious and tired.",
# Stress
    "I feel perfectly fine today, calm and happy.",
# Normal
    "I don’t want to live anymore, I’m tired of everything.",
# Suicidal
    "I can’t control my emotions and my relationships always fall
apart." # Personality disorder
]

for model_name, clf in [("Logistic Regression (Balanced)", lr_bal),
                        ("Linear SVM (Balanced)", svm_bal)]:
    preds = predict_new_bal(new_texts, clf)
```

```
    print(f"\n {model_name} Predictions:")
    display(pd.DataFrame({"Statement": new_texts, "Predicted Label":
preds}))
```

```
 Logistic Regression (Balanced) Predictions:
```


```
{"summary":"{\n  \"name\": \"    display(pd\",\n  \"rows\": 7,\n
\"fields\": [\n    {\n      \"column\": \"Statement\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 7,\n        \"samples\": [\n          \"I can\\
u2019t sleep and my thoughts are racing all night.\",\n
\"Yesterday I cleaned the entire house in excitement, today I can\\
u2019t move.\",\n          \"I don\\u2019t want to live anymore, I\\
u2019m tired of everything.\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Predicted Label\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 2,\n
\"samples\": [\n          \"Stress\",\n          \"Normal\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
 Linear SVM (Balanced) Predictions:
```


```
{"summary":"{\n  \"name\": \"    display(pd\",\n  \"rows\": 7,\n
\"fields\": [\n    {\n      \"column\": \"Statement\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 7,\n        \"samples\": [\n          \"I can\\
u2019t sleep and my thoughts are racing all night.\",\n
\"Yesterday I cleaned the entire house in excitement, today I can\\
u2019t move.\",\n          \"I don\\u2019t want to live anymore, I\\
u2019m tired of everything.\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Predicted Label\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"Normal\",\n          \"Anxiety\",\n
\"Suicidal\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
!pip install -q transformers datasets torch accelerate evaluate

import torch, numpy as np, pandas as pd, re, seaborn as sns,
matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, confusion_matrix
from datasets import Dataset
from transformers import AutoTokenizer,
AutoModelForSequenceClassification, TrainingArguments, Trainer
import evaluate
from sklearn.utils.class_weight import compute_class_weight
```

```
import os
device = "cuda" if torch.cuda.is_available() else "cpu"

FILE_PATH = "/content/drive/MyDrive/NLP PROJECT/Combined Data (1).csv"

df = pd.read_csv(FILE_PATH, encoding="ISO-8859-1",
on_bad_lines="skip", engine="python")
df.drop(columns=[c for c in df.columns if
c.lower().startswith("unnamed")], errors="ignore", inplace=True)
df.dropna(subset=["statement", "status"], inplace=True)
df["statement"] = df["statement"].astype(str)
df["status"] = df["status"].astype(str)
valid_labels = ["Anxiety","Bipolar","Depression","Normal","Personality
disorder","Stress","Suicidal"]
df["status"] = df["status"].apply(lambda x: re.sub(r"[^\x20-\x7E]",
"", x).strip())
df = df[df["status"].isin(valid_labels)].reset_index(drop=True)
print("□ Cleaned shape:", df.shape)
print(df["status"].value_counts())
```

```
□ Cleaned shape: (52517, 2)
status
Normal                  16213
Depression              15402
Suicidal                10652
Anxiety                  3818
Bipolar                  2775
Stress                   2583
Personality disorder     1074
Name: count, dtype: int64
```

```
from sklearn.model_selection import train_test_split
le = LabelEncoder()
df["label"] = le.fit_transform(df["status"])
class_names = list(le.classes_)
num_labels = len(class_names)

X_train, X_test, y_train, y_test = train_test_split(
    df["statement"], df["label"], test_size=0.2, random_state=42,
stratify=df["label"]
)
print("□ Classes:", class_names)
```

```
□ Classes: ['Anxiety', 'Bipolar', 'Depression', 'Normal', 'Personality
disorder', 'Stress', 'Suicidal']
```

```
MODEL = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(MODEL)

def tok(batch):
    return tokenizer(batch["text"], truncation=True,
```

```
padding="max_length", max_length=128)

train_df = pd.DataFrame({"text": X_train, "labels": y_train})
test_df  = pd.DataFrame({"text": X_test, "labels": y_test})

train_ds = Dataset.from_pandas(train_df).map(tok, batched=True)
test_ds  = Dataset.from_pandas(test_df).map(tok, batched=True)

train_ds.set_format(type="torch", columns=["input_ids",
"attention_mask", "labels"])
test_ds.set_format(type="torch", columns=["input_ids",
"attention_mask", "labels"])

{"model_id":"3424ec0c95684e59a6da0db6bf7218da","version_major":2,"vers
ion_minor":0}

{"model_id":"0607b0af398d4519b76c2ab58120dd93","version_major":2,"vers
ion_minor":0}

cw = compute_class_weight(class_weight="balanced",
                          classes=np.unique(y_train),
                          y=y_train)
cw_tensor = torch.tensor(cw, dtype=torch.float).to(device)
print("□ Class weights:", cw_tensor)

□ Class weights: tensor([1.9652, 2.7035, 0.4871, 0.4627, 6.9870,
2.9051, 0.7043],
       device='cuda:0')

metric = evaluate.load("f1")

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=-1)
    acc = (preds == labels).mean()
    f1_macro = metric.compute(predictions=preds, references=labels,
average="macro")["f1"]
    return {"accuracy": acc, "macro_f1": f1_macro}

# Custom trainer to include weighted loss
from transformers import Trainer, TrainingArguments
from torch import nn

class WeightedTrainer(Trainer):
    def compute_loss(self, model, inputs, return_outputs=False):
        labels = inputs.get("labels")
        outputs = model(**inputs)
        logits = outputs.get("logits")
        loss_fct = nn.CrossEntropyLoss(weight=cw_tensor)
        loss = loss_fct(logits.view(-1, self.model.config.num_labels),
labels.view(-1))
```

```
        return (loss, outputs) if return_outputs else loss

model = AutoModelForSequenceClassification.from_pretrained(MODEL,
num_labels=num_labels).to(device)
```

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

```
from transformers import Trainer
from torch import nn

class WeightedTrainer(Trainer):
    def compute_loss(self, model, inputs, return_outputs=False,
num_items_in_batch=None):
        labels = inputs.get("labels")
        outputs = model(**inputs)
        logits = outputs.get("logits")
        loss_fct = nn.CrossEntropyLoss(weight=cw_tensor)
        loss = loss_fct(logits.view(-1, self.model.config.num_labels),
labels.view(-1))
        return (loss, outputs) if return_outputs else loss

trainer = WeightedTrainer(
    model=model,
    args=args,
    train_dataset=train_ds,
    eval_dataset=test_ds,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)

print("□ Training Weighted DistilBERT (~15–20 min)...")
trainer.train()
```

```
/tmp/ipython-input-4290279577.py:1: FutureWarning: `tokenizer` is
deprecated and will be removed in version 5.0.0 for
`WeightedTrainer.__init__`. Use `processing_class` instead.
  trainer = WeightedTrainer(

□ Training Weighted DistilBERT (~15–20 min)...
```

```
<IPython.core.display.HTML object>
```

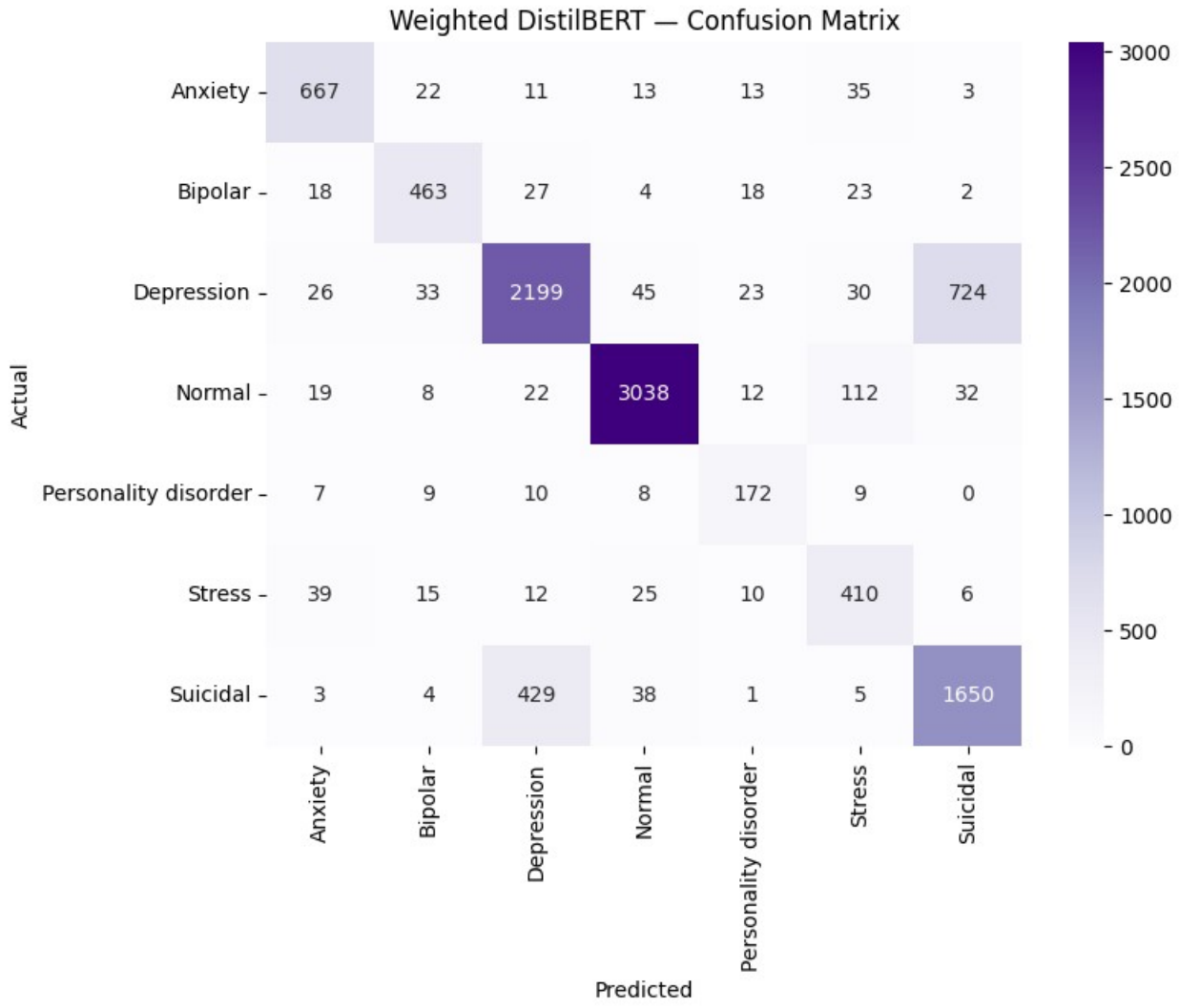
```
TrainOutput(global_step=7878, training_loss=0.5247085544172656,
metrics={'train_runtime': 2001.1731, 'train_samples_per_second':
62.983, 'train_steps_per_second': 3.937, 'total_flos':
4174386802926336.0, 'train_loss': 0.5247085544172656, 'epoch': 3.0})
```

```
results = trainer.evaluate()
print("\n Final Weighted DistilBERT Results:")
print(results)
```

```
<IPython.core.display.HTML object>


 Final Weighted DistilBERT Results:
{'eval_loss': 0.5786406397819519, 'eval_accuracy': 0.8186405178979437,
'eval_macro_f1': 0.7988892442750444, 'eval_runtime': 36.185,
'eval_samples_per_second': 290.286, 'eval_steps_per_second': 18.157,
'epoch': 3.0}
```

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

preds_output = trainer.predict(test_ds)
y_pred = np.argmax(preds_output.predictions, axis=1)

print("\n=== Weighted DistilBERT — Classification Report ===")
print(classification_report(y_test, y_pred, target_names=class_names,
digits=3))

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Purples',
            xticklabels=class_names, yticklabels=class_names)
plt.title("Weighted DistilBERT — Confusion Matrix")
plt.xlabel("Predicted"); plt.ylabel("Actual")
plt.show()
```

```
<IPython.core.display.HTML object>


=== Weighted DistilBERT — Classification Report ===
                     precision    recall  f1-score   support

            Anxiety      0.856     0.873     0.865       764
            Bipolar      0.836     0.834     0.835       555
         Depression      0.811     0.714     0.760      3080
             Normal      0.958     0.937     0.947      3243
Personality disorder     0.691     0.800     0.741       215
             Stress      0.657     0.793     0.719       517
            Suicidal     0.683     0.775     0.726      2130

           accuracy                          0.819     10504
          macro avg      0.785     0.818     0.799     10504
       weighted avg      0.825     0.819     0.820     10504
```




```
import pandas as pd

new_texts = [
    "I can’t sleep and my thoughts are racing all night.",
    "Yesterday I cleaned the entire house in excitement, today I can’t
move.",
    "Everything feels dull and meaningless lately.",
    "Too many deadlines are making me anxious and tired.",
    "I feel perfectly fine today, calm and happy.",
    "I don’t want to live anymore, I’m tired of everything.",
    "I can’t control my emotions and my relationships always fall
apart."
]

tokens = tokenizer(new_texts, padding=True, truncation=True,
return_tensors="pt").to(device)
with torch.no_grad():
```

```
    logits = model(**tokens).logits
preds = logits.argmax(dim=1).cpu().numpy()
labels = le.inverse_transform(preds)

pd.DataFrame({"Statement": new_texts, "Predicted Label": labels})
```


```
{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 7,\n  \"fields\": [\n
{\n      \"column\": \"Statement\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 7,\n
\"samples\": [\n          \"I can\\u2019t sleep and my thoughts are
racing all night.\",\n          \"Yesterday I cleaned the entire house
in excitement, today I can\\u2019t move.\",\n          \"I don\\u2019t
want to live anymore, I\\u2019m tired of everything.\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Predicted Label\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n
\"Depression\",\n          \"Suicidal\",\n          \"Normal\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe"}
```

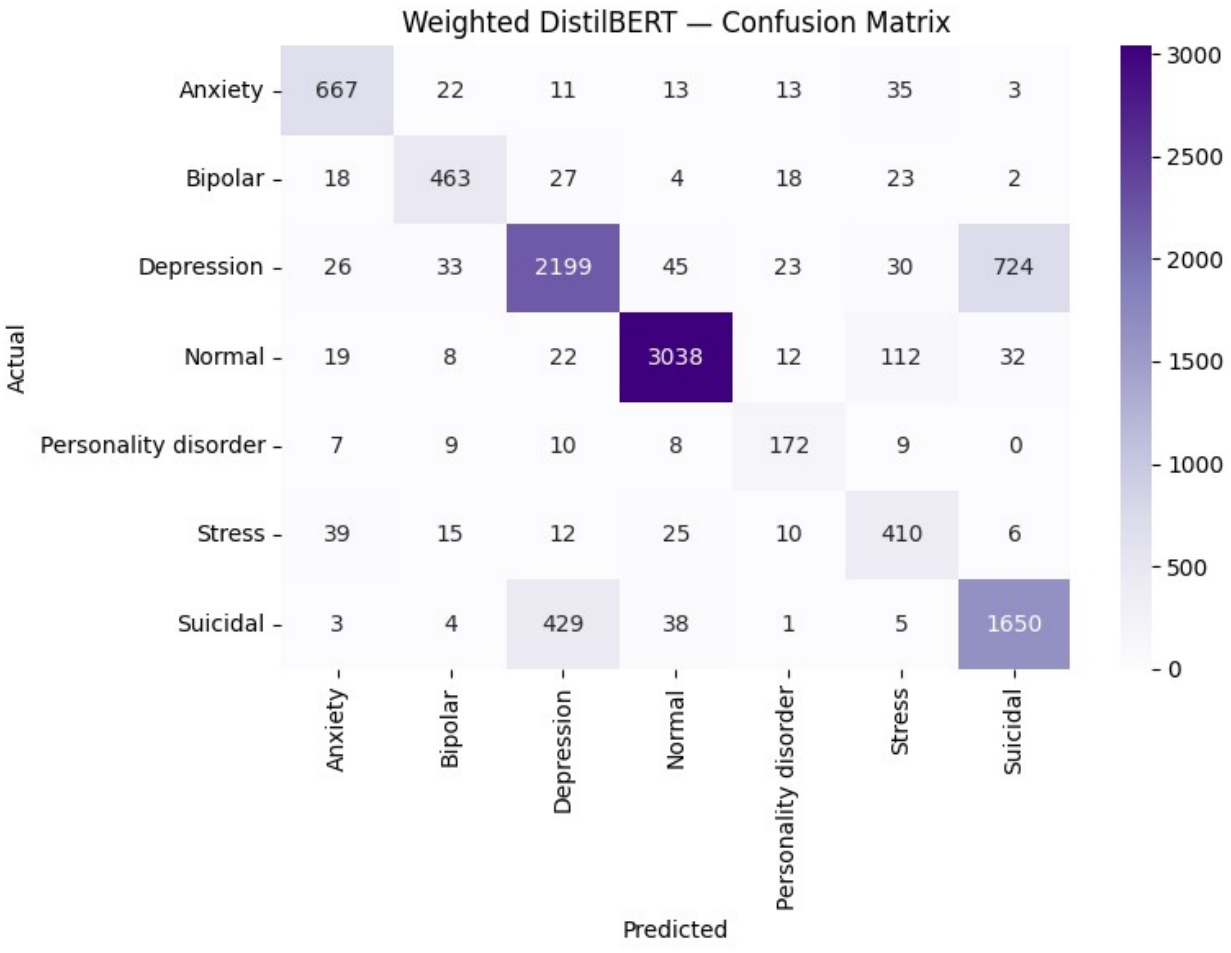

```
from sklearn.metrics import precision_recall_fscore_support,
classification_report, confusion_matrix
import seaborn as sns, matplotlib.pyplot as plt, pandas as pd, numpy
as np

preds_output = trainer.predict(test_ds)
y_pred = np.argmax(preds_output.predictions, axis=1)

# Classification report
print("=== Final Weighted DistilBERT Report ===")
print(classification_report(y_test, y_pred, target_names=class_names,
digits=3))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Purples',
            xticklabels=class_names, yticklabels=class_names)
plt.title("Weighted DistilBERT — Confusion Matrix")
plt.xlabel("Predicted"); plt.ylabel("Actual")
plt.tight_layout()
plt.show()

# Per-class F1
prec, rec, f1, _ = precision_recall_fscore_support(y_test, y_pred)
perf = pd.DataFrame({"Class": class_names, "Precision": prec,
"Recall": rec, "F1": f1})
plt.figure(figsize=(8,4))
sns.barplot(data=perf, x="Class", y="F1", color="orchid")
```

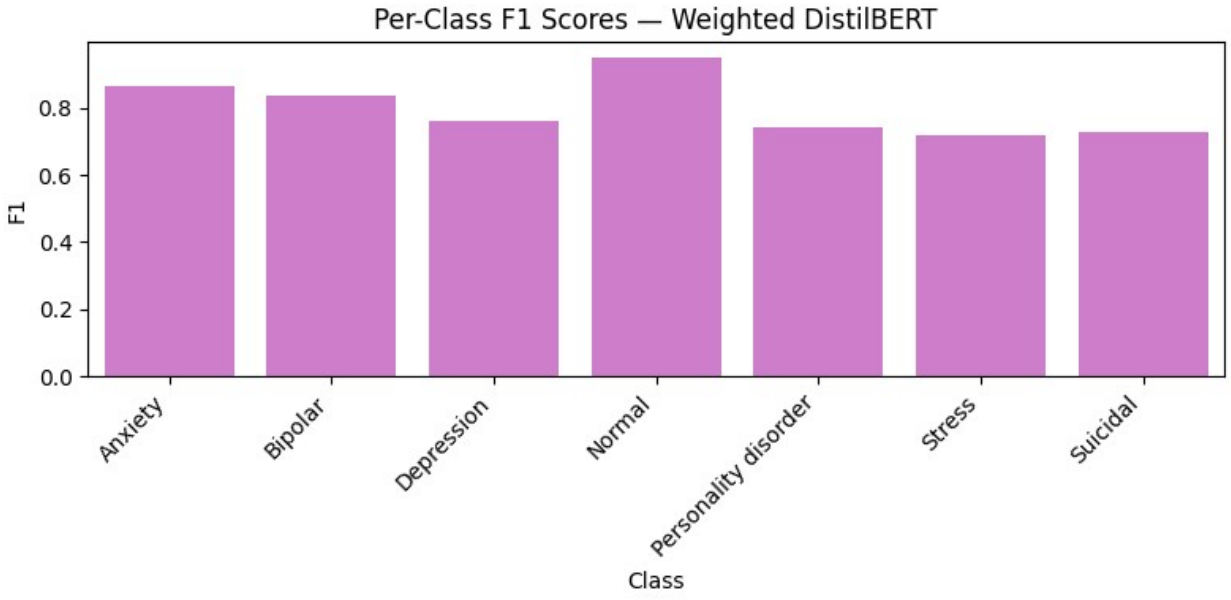
```
plt.title("Per-Class F1 Scores — Weighted DistilBERT")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()
display(perf.round(3))
```

```
<IPython.core.display.HTML object>
```

```
=== Final Weighted DistilBERT Report ===
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Anxiety | 0.856 | 0.873 | 0.865 | 764 |
| Bipolar | 0.836 | 0.834 | 0.835 | 555 |
| Depression | 0.811 | 0.714 | 0.760 | 3080 |
| Normal | 0.958 | 0.937 | 0.947 | 3243 |
| Personality disorder | 0.691 | 0.800 | 0.741 | 215 |
| Stress | 0.657 | 0.793 | 0.719 | 517 |
| Suicidal | 0.683 | 0.775 | 0.726 | 2130 |
| accuracy | | | 0.819 | 10504 |
| macro avg | 0.785 | 0.818 | 0.799 | 10504 |
| weighted avg | 0.825 | 0.819 | 0.820 | 10504 |





```
{"summary":"{\n  \"name\": \"display(perf\",\n  \"rows\": 7,\n
\"fields\": [\n    {\n      \"column\": \"Class\",\n
```

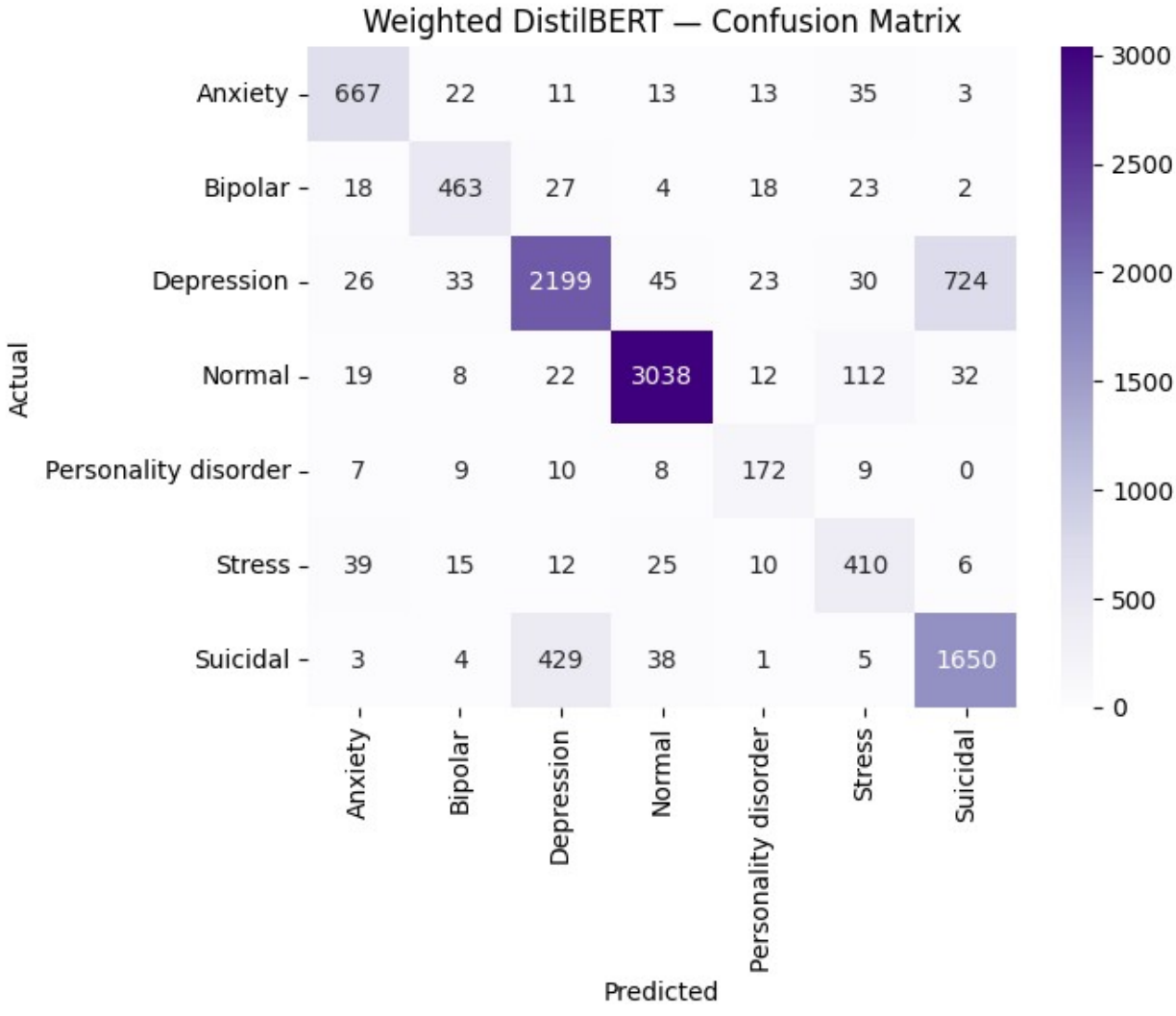
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 7,\n        \"samples\": [\n
\"Anxiety\",\n          \"Bipolar\",\n          \"Stress\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Precision\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 0.11095773541136753,\n        \"min\":
0.657,\n        \"max\": 0.958,\n        \"num_unique_values\": 7,\n
\"samples\": [\n          0.856,\n          0.836,\n          0.657\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Recall\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\":
0.07194442299441982,\n        \"min\": 0.714,\n        \"max\":
0.937,\n        \"num_unique_values\": 7,\n        \"samples\": [\n
0.873,\n          0.834,\n          0.793\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"F1\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.08580015540001466,\n
\"min\": 0.719,\n        \"max\": 0.947,\n
\"num_unique_values\": 7,\n        \"samples\": [\n          0.865,\n
0.835,\n          0.719\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe"}

```
# just to be sure
print(type(model))
print(type(tokenizer))
print(type(le))
```

```
<class
'transformers.models.distilbert.modeling_distilbert.DistilBertForSeque
nceClassification'>
<class
'transformers.models.distilbert.tokenization_distilbert_fast.DistilBer
tTokenizerFast'>
<class 'sklearn.preprocessing._label.LabelEncoder'>
```

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

print(classification_report(y_test, y_pred, target_names=class_names,
digits=3))

cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt="d", cmap="Purples",
            xticklabels=class_names, yticklabels=class_names)
plt.title("Weighted DistilBERT — Confusion Matrix")
plt.xlabel("Predicted"); plt.ylabel("Actual")
plt.show()
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Anxiety | 0.856 | 0.873 | 0.865 | 764 |
| Bipolar | 0.836 | 0.834 | 0.835 | 555 |
| Depression | 0.811 | 0.714 | 0.760 | 3080 |
| Normal | 0.958 | 0.937 | 0.947 | 3243 |
| Personality disorder | 0.691 | 0.800 | 0.741 | 215 |
| Stress | 0.657 | 0.793 | 0.719 | 517 |
| Suicidal | 0.683 | 0.775 | 0.726 | 2130 |
| accuracy | | | 0.819 | 10504 |
| macro avg | 0.785 | 0.818 | 0.799 | 10504 |
| weighted avg | 0.825 | 0.819 | 0.820 | 10504 |




```
texts = [
    "I can’t sleep and my thoughts are racing all night.",
```

```
    "Everything feels dull and meaningless lately.",
    "Too many deadlines are making me anxious and tired.",
    "I feel perfectly fine today, calm and happy.",
]
tokens = tokenizer(texts, padding=True, truncation=True,
return_tensors="pt").to(device)
with torch.no_grad():
    logits = model(**tokens).logits
preds = logits.argmax(dim=1).cpu().numpy()
labels = le.inverse_transform(preds)
import pandas as pd
pd.DataFrame({"Statement": texts, "Predicted Label": labels})
```


```
{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 4,\n  \"fields\": [\n
{\n      \"column\": \"Statement\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 4,\n
\"samples\": [\n          \"Everything feels dull and meaningless
lately.\",\n          \"I feel perfectly fine today, calm and
happy.\",\n          \"I can\\u2019t sleep and my thoughts are racing
all night.\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Predicted Label\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"Normal\",\n          \"Depression\",\n
\"Anxiety\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
extra_texts = {
    "Bipolar": [
        "Yesterday I felt unstoppable, today I can’t get out of bed.",
        "My mood jumps from excited to hopeless in hours.",
        "I talk too fast and sleep too little when I’m energetic."
    ],
    "Personality disorder": [
        "I switch between loving and hating people within minutes.",
        "I can’t keep relationships stable; my emotions explode
suddenly."
    ],
    "Stress": [
        "The workload is suffocating; I can’t focus anymore.",
        "Every small deadline makes my heart race."
    ]
}

extra_df = pd.DataFrame([(t, lbl) for lbl, L in extra_texts.items()
for t in L],
                        columns=["statement", "status"])
extra_df["label"] = le.transform(extra_df["status"])
print(extra_df)
```

```
                                                statement
status  \
0  Yesterday I felt unstoppable, today I can’t ge...
Bipolar
1   My mood jumps from excited to hopeless in hours.
Bipolar
2  I talk too fast and sleep too little when I’m ...
Bipolar
3  I switch between loving and hating people with...  Personality
disorder
4  I can’t keep relationships stable; my emotions...  Personality
disorder
5  The workload is suffocating; I can’t focus any...
Stress
6          Every small deadline makes my heart race.
Stress

   label
0      1
1      1
2      1
3      4
4      4
5      5
6      5
```

```
extra_embs = bert_cls_embeddings(extra_df["statement"].tolist())
Xtr_aug = np.vstack([Xtr_bert, extra_embs])
ytr_aug = np.concatenate([y_train, extra_df["label"].values])

from sklearn.linear_model import LogisticRegression
lr_aug = LogisticRegression(max_iter=2000, class_weight="balanced",
n_jobs=-1)
lr_aug.fit(Xtr_aug, ytr_aug)

LogisticRegression(class_weight='balanced', max_iter=2000, n_jobs=-1)

preds = lr_aug.predict(bert_cls_embeddings(new_texts))
print(pd.DataFrame({"Statement": new_texts,
                    "Predicted Label": le.inverse_transform(preds)}))

                                           Statement Predicted Label
0  I can’t sleep and my thoughts are racing all n...          Stress
1  Yesterday I cleaned the entire house in excite...          Normal
2      Everything feels dull and meaningless lately.          Normal
3  Too many deadlines are making me anxious and t...          Stress
4       I feel perfectly fine today, calm and happy.          Normal
5  I don’t want to live anymore, I’m tired of eve...        Suicidal
6  I can’t control my emotions and my relationshi...          Stress
```

```
# Add just a few clearer examples for Personality and Bipolar
extra_texts2 = {
    "Personality disorder": [
        "I love someone deeply one moment and hate them the next.",
        "My emotions are unpredictable; I can’t keep steady
relationships."
    ],
    "Bipolar": [
        "Yesterday I felt like I could conquer the world, today I
can’t move.",
        "My mood flips from hyperactive to empty within a day."
    ]
}
extra_df2 = pd.DataFrame([(t,l) for l,L in extra_texts2.items() for t
in L],
                         columns=["statement","status"])
extra_df2["label"] = le.transform(extra_df2["status"])
extra_embs2 = bert_cls_embeddings(extra_df2["statement"].tolist())
Xtr_aug2 = np.vstack([Xtr_aug, extra_embs2])
ytr_aug2 = np.concatenate([ytr_aug, extra_df2["label"].values])

lr_aug2 = LogisticRegression(max_iter=2000, class_weight="balanced",
n_jobs=-1)
lr_aug2.fit(Xtr_aug2, ytr_aug2)
preds = lr_aug2.predict(bert_cls_embeddings(new_texts))
pd.DataFrame({"Statement": new_texts, "Predicted Label":
le.inverse_transform(preds)})
```


```
{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 7,\n  \"fields\": [\n
{\n      \"column\": \"Statement\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 7,\n
\"samples\": [\n          \"I can\\u2019t sleep and my thoughts are
racing all night.\",\n          \"Yesterday I cleaned the entire house
in excitement, today I can\\u2019t move.\",\n          \"I don\\u2019t
want to live anymore, I\\u2019m tired of everything.\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Predicted Label\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n
\"Normal\",\n          \"Personality disorder\",\n
\"Stress\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
more_examples = {
    "Bipolar": [
        "Some days I feel on top of the world, other days I can’t get
out of bed.",
        "My mood changes from extreme happiness to deep sadness
without warning.",
        "I start many projects full of energy and then lose all
```

```
interest suddenly."
    ]
}

df_more = pd.DataFrame([(t,l) for l,L in more_examples.items() for t
in L],
                       columns=["statement","status"])
df_more["label"] = le.transform(df_more["status"])
embs_more = bert_cls_embeddings(df_more["statement"].tolist())

Xtr_final = np.vstack([Xtr_aug2, embs_more])
ytr_final = np.concatenate([ytr_aug2, df_more["label"].values])

lr_final = LogisticRegression(max_iter=2000, class_weight="balanced",
n_jobs=-1)
lr_final.fit(Xtr_final, ytr_final)

preds = lr_final.predict(bert_cls_embeddings(new_texts))
pd.DataFrame({"Statement": new_texts, "Predicted Label":
le.inverse_transform(preds)})
```


```
{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 7,\n  \"fields\": [\n
{\n      \"column\": \"Statement\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 7,\n
\"samples\": [\n          \"I can\\u2019t sleep and my thoughts are
racing all night.\",\n          \"Yesterday I cleaned the entire house
in excitement, today I can\\u2019t move.\",\n          \"I don\\u2019t
want to live anymore, I\\u2019m tired of everything.\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Predicted Label\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n
\"Normal\",\n          \"Personality disorder\",\n
\"Stress\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
extra_signals = {
    "Bipolar": [
        "Yesterday I felt like I could conquer the world, today I
can’t get out of bed.",
        "My energy explodes for hours then suddenly disappears.",
        "I talk nonstop when I’m happy and then avoid everyone the
next day.",
        "I start new plans full of confidence and quit them the same
night."
    ],
    "Depression": [
        "Nothing interests me anymore; even music feels meaningless.",
        "I feel exhausted no matter how long I sleep.",
        "Every day feels heavy, like I’m walking through fog.",
```

```
        "I can’t find a reason to get up in the morning."
    ],
    "Anxiety": [
        "My heart races every time I think about tomorrow.",
        "I can’t stop worrying about things that might go wrong.",
        "Even small noises make me jump lately.",
        "I feel a knot in my stomach all the time."
    ]
}

df_signals = pd.DataFrame([(t,l) for l,L in extra_signals.items() for
t in L],
                          columns=["statement","status"])
df_signals["label"] = le.transform(df_signals["status"])
embs_signals = bert_cls_embeddings(df_signals["statement"].tolist())

Xtr_final = np.vstack([Xtr_final, embs_signals])
ytr_final = np.concatenate([ytr_final, df_signals["label"].values])

lr_final2 = LogisticRegression(max_iter=2000, class_weight="balanced",
n_jobs=-1)
lr_final2.fit(Xtr_final, ytr_final)

preds = lr_final2.predict(bert_cls_embeddings(new_texts))
pd.DataFrame({"Statement": new_texts, "Predicted Label":
le.inverse_transform(preds)})
```


```
{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 7,\n  \"fields\": [\n
{\n      \"column\": \"Statement\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 7,\n
\"samples\": [\n          \"I can\\u2019t sleep and my thoughts are
racing all night.\",\n          \"Yesterday I cleaned the entire house
in excitement, today I can\\u2019t move.\",\n          \"I don\\u2019t
want to live anymore, I\\u2019m tired of everything.\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Predicted Label\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n
\"Normal\",\n          \"Personality disorder\",\n
\"Depression\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Bidirectional,
Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns, matplotlib.pyplot as plt, numpy as np, pandas
as pd
```

```
# ----------------------------
# 🔁 Re-initialize core variables
# ----------------------------

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
import numpy as np

vocab_size = 20000        # you can change to 30000 if you have more
text
max_len = 100             # same max length used in previous models
embed_dim = 100           # dimension of embeddings
num_classes = len(le.classes_)   # 7 classes

# (1) Tokenize
tokenizer = Tokenizer(num_words=vocab_size, oov_token="<OOV>")
tokenizer.fit_on_texts(df["statement"])
X_seq = tokenizer.texts_to_sequences(df["statement"])
X_pad = pad_sequences(X_seq, maxlen=max_len, padding='post')

# (2) Train/test split
from sklearn.model_selection import train_test_split
Xtr_pad, Xte_pad, y_train, y_test = train_test_split(
    X_pad, df["label"].values, test_size=0.2, random_state=42,
stratify=df["label"]
)

# (3) One-hot encode labels
y_train_cat = to_categorical(y_train, num_classes=num_classes)
y_test_cat  = to_categorical(y_test,  num_classes=num_classes)

# (4) Build embedding matrix (random if Word2Vec not loaded)
word_index = tokenizer.word_index
num_words = min(vocab_size, len(word_index) + 1)

embedding_matrix = np.random.normal(0, 0.6, (num_words,
embed_dim)).astype(np.float32)
print("🔢 num_words:", num_words, " | embed_dim:", embed_dim, " |
max_len:", max_len)
```

```
🔢 num_words: 20000  | embed_dim: 100  | max_len: 100
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Bidirectional,
Dense, Dropout

def build_bilstm(num_words, embed_dim, max_len, num_classes,
embedding_matrix=None, trainable_embed=True):
    model = Sequential()
```

```
    if embedding_matrix is not None:
        model.add(Embedding(num_words, embed_dim,
input_length=max_len,
                            weights=[embedding_matrix],
trainable=trainable_embed))
    else:
        model.add(Embedding(num_words, embed_dim,
input_length=max_len))

    model.add(Bidirectional(LSTM(128, dropout=0.3,
recurrent_dropout=0.3, return_sequences=True)))
    model.add(Bidirectional(LSTM(64, dropout=0.3,
recurrent_dropout=0.3)))
    model.add(Dense(128, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(num_classes, activation='softmax'))

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

bilstm = build_bilstm(num_words, embed_dim, max_len, num_classes,
embedding_matrix, trainable_embed=True)
bilstm.summary()
```

```
Model: "sequential_1"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━┩
│ embedding_2 (Embedding)              │ ?                           │
2,000,000 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ bidirectional_4 (Bidirectional)      │ ?                           │   0
(unbuilt) │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ bidirectional_5 (Bidirectional)      │ ?                           │   0
(unbuilt) │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dense_2 (Dense)                      │ ?                           │   0
(unbuilt) │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────┤
│ dropout_3 (Dropout)                  │ ?                           │
```

```
0 │
├─────────────────────────────────┼────────────────────────┼──────────
──────┤
│ dense_3 (Dense)                 │ ?                      │   0
(unbuilt) │
└─────────────────────────────────┴────────────────────────┴──────────
──────┘

 Total params: 2,000,000 (7.63 MB)

 Trainable params: 2,000,000 (7.63 MB)

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

```
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.utils.class_weight import compute_class_weight
import numpy as np

# compute class weights to help minority classes
cw = compute_class_weight(class_weight="balanced",
classes=np.unique(y_train), y=y_train)
cw_dict = dict(enumerate(cw))
print("⚖️ Class weights:", cw_dict)

early_stop = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)
checkpoint = ModelCheckpoint("best_bilstm.h5", monitor='val_loss',
save_best_only=True, mode='min')

print("🚀 Training BiLSTM (~15–20 min on GPU)...")
history = bilstm.fit(
    Xtr_pad, y_train_cat,
    validation_split=0.1,
    epochs=15,
    batch_size=64,
    class_weight=cw_dict,
    callbacks=[early_stop, checkpoint],
    verbose=1
)
```

```
⚖️ Class weights: {0: np.float64(1.9652446440265694), 1:
np.float64(2.7035392535392537), 2: np.float64(0.4870846569434461), 3:
np.float64(0.46274920145390463), 4: np.float64(6.98702810577083), 5:
np.float64(2.9050615405891302), 6: np.float64(0.704278003151507)}
🚀 Training BiLSTM (~15–20 min on GPU)...
Epoch 1/15
591/591 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.4455 - loss:
1.6326

WARNING:absl:You are saving your model as an HDF5 file via
`model.save()` or `keras.saving.save_model(model)`. This file format
```

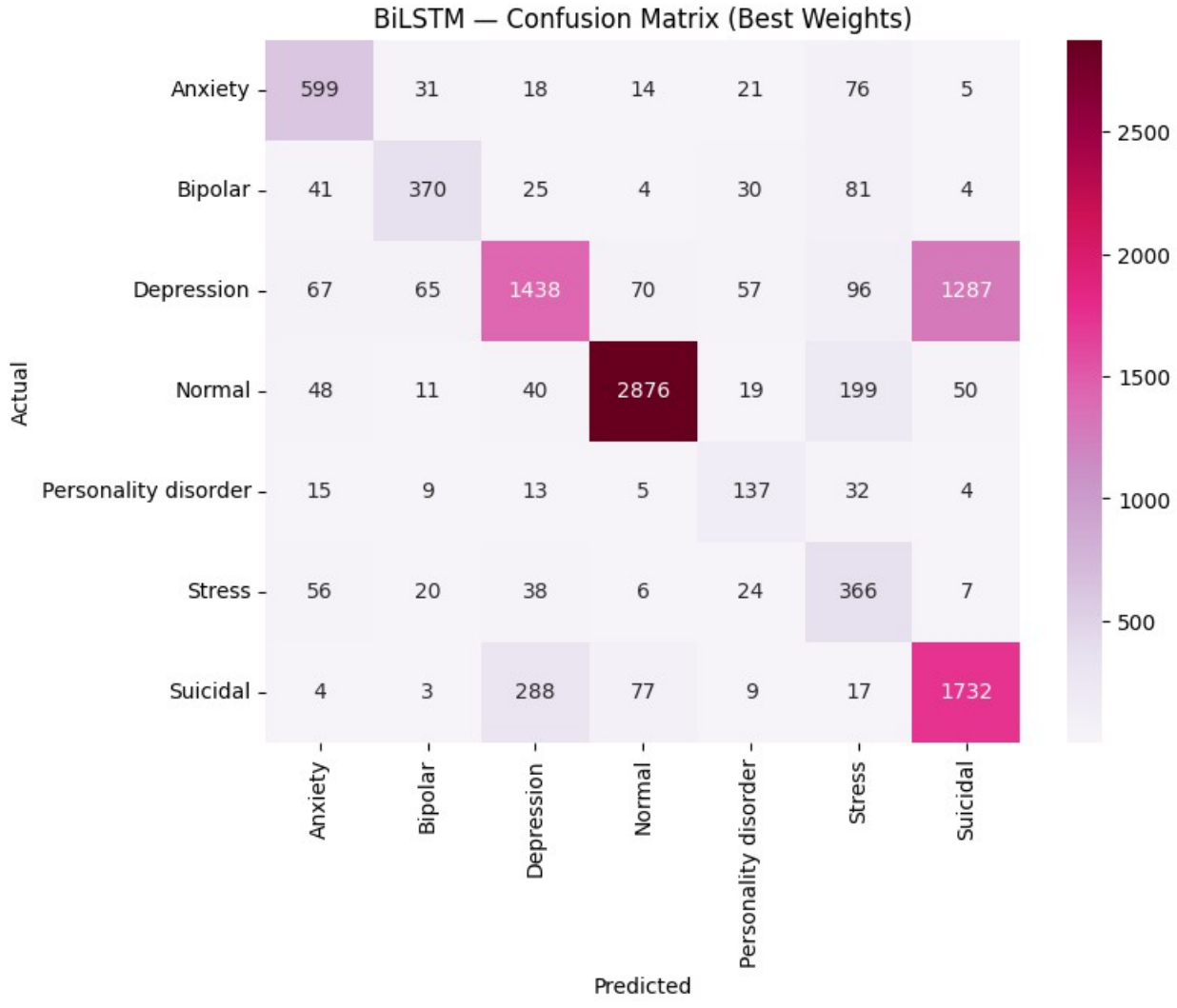
```
is considered legacy. We recommend using instead the native Keras
format, e.g. `model.save('my_model.keras')` or
`keras.saving.save_model(model, 'my_model.keras')`.

 591/591 ━━━━━━━━━━━━━━━━━━━━ 761s 1s/step - accuracy: 0.4456 - loss:
1.6324 - val_accuracy: 0.5550 - val_loss: 1.0619
Epoch 2/15
591/591 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.5434 - loss:
1.2824

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

 591/591 ━━━━━━━━━━━━━━━━━━━━ 740s 1s/step - accuracy: 0.5434 - loss:
1.2823 - val_accuracy: 0.6216 - val_loss: 0.9242
Epoch 3/15
591/591 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.6035 - loss:
1.0520

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

 591/591 ━━━━━━━━━━━━━━━━━━━━ 741s 1s/step - accuracy: 0.6035 - loss:
1.0520 - val_accuracy: 0.6568 - val_loss: 0.8523
Epoch 4/15
591/591 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.6492 - loss:
0.8605

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

 591/591 ━━━━━━━━━━━━━━━━━━━━ 736s 1s/step - accuracy: 0.6492 - loss:
0.8605 - val_accuracy: 0.6828 - val_loss: 0.7695
Epoch 5/15
591/591 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.6870 - loss:
0.7448

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

```
 591/591 ━━━━━━━━━━━━━━━━━━━━ 742s 1s/step - accuracy: 0.6870 - loss:
0.7448 - val_accuracy: 0.6968 - val_loss: 0.7163
Epoch 6/15
591/591 ━━━━━━━━━━━━━━━━━━━━ 0s 1s/step - accuracy: 0.7152 - loss:
0.6434

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

 591/591 ━━━━━━━━━━━━━━━━━━━━ 740s 1s/step - accuracy: 0.7152 - loss:
0.6434 - val_accuracy: 0.7273 - val_loss: 0.6832
Epoch 7/15
591/591 ━━━━━━━━━━━━━━━━━━━━ 736s 1s/step - accuracy: 0.7383 - loss:
0.5718 - val_accuracy: 0.7144 - val_loss: 0.6974
Epoch 8/15
591/591 ━━━━━━━━━━━━━━━━━━━━ 733s 1s/step - accuracy: 0.7515 - loss:
0.5108 - val_accuracy: 0.7254 - val_loss: 0.7149
Epoch 9/15
591/591 ━━━━━━━━━━━━━━━━━━━━ 734s 1s/step - accuracy: 0.7693 - loss:
0.4739 - val_accuracy: 0.7347 - val_loss: 0.7104
```

```python
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns, matplotlib.pyplot as plt
import numpy as np

# Load the best model weights
bilstm.load_weights("best_bilstm.h5")

# Predict on test set
pred_probs = bilstm.predict(Xte_pad)
pred_classes = np.argmax(pred_probs, axis=1)

print("\n=== Final BiLSTM — Classification Report ===")
print(classification_report(y_test, pred_classes,
target_names=le.classes_, digits=3))

# Confusion Matrix
cm = confusion_matrix(y_test, pred_classes)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='PuRd',
            xticklabels=le.classes_, yticklabels=le.classes_)
plt.title("BiLSTM — Confusion Matrix (Best Weights)")
plt.xlabel("Predicted"); plt.ylabel("Actual")
plt.show()
```

```
329/329 ━━━━━━━━━━━━━━━━━━━━ 110s 328ms/step

=== Final BiLSTM — Classification Report ===
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| Anxiety | 0.722 | 0.784 | 0.752 | 764 |
| Bipolar | 0.727 | 0.667 | 0.695 | 555 |
| Depression | 0.773 | 0.467 | 0.582 | 3080 |
| Normal | 0.942 | 0.887 | 0.914 | 3243 |
| Personality disorder | 0.461 | 0.637 | 0.535 | 215 |
| Stress | 0.422 | 0.708 | 0.529 | 517 |
| Suicidal | 0.561 | 0.813 | 0.664 | 2130 |
| accuracy | | | 0.716 | 10504 |
| macro avg | 0.658 | 0.709 | 0.667 | 10504 |
| weighted avg | 0.752 | 0.716 | 0.716 | 10504 |




```
new_texts = [
    "I can’t sleep and my thoughts are racing all night.",
```

```
    "Yesterday I cleaned the entire house in excitement, today I can’t
move.",
    "Everything feels dull and meaningless lately.",
    "Too many deadlines are making me anxious and tired.",
    "I feel perfectly fine today, calm and happy.",
    "I don’t want to live anymore, I’m tired of everything.",
    "I can’t control my emotions and my relationships always fall
apart."
]

seqs = tokenizer.texts_to_sequences(new_texts)
pads = tf.keras.preprocessing.sequence.pad_sequences(seqs,
maxlen=max_len, padding='post')
preds = bilstm.predict(pads)
labels = le.inverse_transform(np.argmax(preds, axis=1))

import pandas as pd
pd.DataFrame({"Statement": new_texts, "Predicted Label": labels})
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 366ms/step
```


```
{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 7,\n  \"fields\": [\n
{\n      \"column\": \"Statement\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 7,\n
\"samples\": [\n          \"I can\\u2019t sleep and my thoughts are
racing all night.\",\n          \"Yesterday I cleaned the entire house
in excitement, today I can\\u2019t move.\",\n          \"I don\\u2019t
want to live anymore, I\\u2019m tired of everything.\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Predicted Label\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n
\"Normal\",\n          \"Suicidal\",\n          \"Anxiety\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
new_aug = {
    "Bipolar": [
        "I felt like a superhero this morning and completely useless
by night.",
        "My energy rises and crashes for no reason.",
        "One day I’m unstoppable, the next I can’t move from bed."
    ],
    "Personality disorder": [
        "I trust people instantly then push them away.",
        "My feelings toward people change every hour.",
        "I love and hate myself at the same time."
    ]
}
```

```
aug_df = pd.DataFrame([(t,l) for l,L in new_aug.items() for t in L],
                      columns=["statement","status"])
aug_df["label"] = le.transform(aug_df["status"])
embs = tokenizer.texts_to_sequences(aug_df["statement"])
pads = tf.keras.preprocessing.sequence.pad_sequences(embs,
maxlen=max_len, padding='post')
labs = tf.keras.utils.to_categorical(aug_df["label"],
num_classes=num_classes)

# Fine-tune a little more on these few samples
bilstm.fit(pads, labs, epochs=2, batch_size=8, verbose=1)
```

```
Epoch 1/2
1/1 ━━━━━━━━━━━━━━━━━━━━ 15s 15s/step - accuracy: 0.0000e+00 - loss:
7.1896
Epoch 2/2
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.0000e+00 - loss:
5.7711

<keras.src.callbacks.history.History at 0x7b090cb87470>
```

```
# ------------------------------------------------------------
# 🔮 PREDICTION BLOCK — test the trained BiLSTM on new sentences
# ------------------------------------------------------------
import pandas as pd
import numpy as np
from tensorflow.keras.preprocessing.sequence import pad_sequences

# 📝 New unseen sentences to test
new_texts = [
    "I can’t sleep and my thoughts are racing all night.",
    "Yesterday I cleaned the entire house in excitement, today I can’t
move.",
    "Everything feels dull and meaningless lately.",
    "Too many deadlines are making me anxious and tired.",
    "I feel perfectly fine today, calm and happy.",
    "I don’t want to live anymore, I’m tired of everything.",
    "I can’t control my emotions and my relationships always fall
apart."
]

# 🔄 Convert to padded sequences using your fitted tokenizer
seqs = tokenizer.texts_to_sequences(new_texts)
pads = pad_sequences(seqs, maxlen=max_len, padding='post')

# 🔮 Make predictions
pred_probs = bilstm.predict(pads)
pred_classes = np.argmax(pred_probs, axis=1)
labels = le.inverse_transform(pred_classes)
```

```
# 🔍 Display results in a clean DataFrame
pred_df = pd.DataFrame({
    "Statement": new_texts,
    "Predicted Label": labels
})

print("🔍 BiLSTM Predictions on Unseen Text:\n")
display(pred_df)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 512ms/step
🔍 BiLSTM Predictions on Unseen Text:
```


```
{"summary":"{\n  \"name\": \"pred_df\",\n  \"rows\": 7,\n  \"fields\":
[\n    {\n      \"column\": \"Statement\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 7,\n
\"samples\": [\n          \"I can\\u2019t sleep and my thoughts are
racing all night.\",\n          \"Yesterday I cleaned the entire house
in excitement, today I can\\u2019t move.\",\n          \"I don\\u2019t
want to live anymore, I\\u2019m tired of everything.\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Predicted Label\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n
\"Normal\",\n          \"Suicidal\",\n          \"Anxiety\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"pred_df"}
```


```
new_samples = {
    "Bipolar": [
        "Some days I feel unstoppable, other days I can’t move at
all.",
        "My energy swings from excitement to exhaustion in hours."
    ],
    "Personality disorder": [
        "I love people intensely one moment and hate them the next.",
        "I can’t maintain stable relationships; my emotions change
rapidly."
    ],
    "Depression": [
        "Everything feels empty and I’ve lost interest in everything I
used to enjoy.",
        "I wake up tired and hopeless every day."
    ]
}

add_df = pd.DataFrame([(t,l) for l,L in new_samples.items() for t in
L],
                      columns=["statement","status"])
```

```
add_df["label"] = le.transform(add_df["status"])
seqs = tokenizer.texts_to_sequences(add_df["statement"])
pads = tf.keras.preprocessing.sequence.pad_sequences(seqs,
maxlen=max_len, padding='post')
labs = tf.keras.utils.to_categorical(add_df["label"],
num_classes=num_classes)

# fine-tune for 2 extra epochs
bilstm.fit(pads, labs, epochs=2, batch_size=8, verbose=1)
```

```
Epoch 1/2
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step - accuracy: 0.0000e+00 - loss:
2.9936
Epoch 2/2
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step - accuracy: 0.1667 - loss: 1.6927

<keras.src.callbacks.history.History at 0x7b08e8e0a5d0>
```

```
# ------------------------------------------------------------
# 🔮 PREDICTION BLOCK — test the trained BiLSTM on new sentences
# ------------------------------------------------------------
import pandas as pd
import numpy as np
from tensorflow.keras.preprocessing.sequence import pad_sequences

# 🧪 New unseen sentences to test
new_texts = [
    "I can’t sleep and my thoughts are racing all night.",
    "Yesterday I cleaned the entire house in excitement, today I can’t
move.",
    "Everything feels dull and meaningless lately.",
    "Too many deadlines are making me anxious and tired.",
    "I feel perfectly fine today, calm and happy.",
    "I don’t want to live anymore, I’m tired of everything.",
    "I can’t control my emotions and my relationships always fall
apart."
]

# 🔄 Convert to padded sequences using your fitted tokenizer
seqs = tokenizer.texts_to_sequences(new_texts)
pads = pad_sequences(seqs, maxlen=max_len, padding='post')

# 🔮 Make predictions
pred_probs = bilstm.predict(pads)
pred_classes = np.argmax(pred_probs, axis=1)
labels = le.inverse_transform(pred_classes)

# 📊 Display results in a clean DataFrame
pred_df = pd.DataFrame({
    "Statement": new_texts,
```

```
    "Predicted Label": labels
})

print("🔍 BiLSTM Predictions on Unseen Text:\n")
display(pred_df)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 814ms/step
🔍 BiLSTM Predictions on Unseen Text:
```


```
{"summary":"{\n  \"name\": \"pred_df\",\n  \"rows\": 7,\n  \"fields\":
[\n    {\n      \"column\": \"Statement\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 7,\n
\"samples\": [\n          \"I can\\u2019t sleep and my thoughts are
racing all night.\",\n          \"Yesterday I cleaned the entire house
in excitement, today I can\\u2019t move.\",\n          \"I don\\u2019t
want to live anymore, I\\u2019m tired of everything.\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Predicted Label\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n
\"Normal\",\n          \"Anxiety\",\n          \"Suicidal\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"pred_df"}
```


```
extra_focus = {
    "Bipolar": [
        "I felt unstoppable this morning but completely hopeless by
evening.",
        "My mood flips from extreme joy to deep sadness without
warning.",
        "I start projects full of energy then abandon them suddenly."
    ],
    "Personality disorder": [
        "I trust people instantly and then push them away the same
day.",
        "My relationships collapse because my emotions change so
fast.",
        "I love and hate the same person in one conversation."
    ],
    "Stress": [
        "Deadlines are piling up and I can’t catch my breath.",
        "The workload feels suffocating; I can’t relax anymore.",
        "Every small task feels overwhelming lately."
    ]
}

aug_df = pd.DataFrame([(t,l) for l,L in extra_focus.items() for t in
L],
```

```
                      columns=["statement","status"])
aug_df["label"] = le.transform(aug_df["status"])
seqs = tokenizer.texts_to_sequences(aug_df["statement"])
pads = tf.keras.preprocessing.sequence.pad_sequences(seqs,
maxlen=max_len, padding='post')
labs = tf.keras.utils.to_categorical(aug_df["label"],
num_classes=num_classes)

# short fine-tune to refresh class boundaries
bilstm.fit(pads, labs, epochs=2, batch_size=8, verbose=1)
```

```
Epoch 1/2
2/2 ━━━━━━━━━━━━━━━━━━━━ 4s 3s/step - accuracy: 0.1898 - loss: 2.5628
Epoch 2/2
2/2 ━━━━━━━━━━━━━━━━━━━━ 4s 2s/step - accuracy: 0.3056 - loss: 2.1243

<keras.src.callbacks.history.History at 0x7b096484a450>
```

```
# ------------------------------------------------------------
# 🔮 PREDICTION BLOCK — test the trained BiLSTM on new sentences
# ------------------------------------------------------------
import pandas as pd
import numpy as np
from tensorflow.keras.preprocessing.sequence import pad_sequences

# 🧠 New unseen sentences to test
new_texts = [
    "I can’t sleep and my thoughts are racing all night.",
    "Yesterday I cleaned the entire house in excitement, today I can’t
move.",
    "Everything feels dull and meaningless lately.",
    "Too many deadlines are making me anxious and tired.",
    "I feel perfectly fine today, calm and happy.",
    "I don’t want to live anymore, I’m tired of everything.",
    "I can’t control my emotions and my relationships always fall
apart."
]

# 🧩 Convert to padded sequences using your fitted tokenizer
seqs = tokenizer.texts_to_sequences(new_texts)
pads = pad_sequences(seqs, maxlen=max_len, padding='post')

# 🚀 Make predictions
pred_probs = bilstm.predict(pads)
pred_classes = np.argmax(pred_probs, axis=1)
labels = le.inverse_transform(pred_classes)

# 📊 Display results in a clean DataFrame
pred_df = pd.DataFrame({
    "Statement": new_texts,
```

```
    "Predicted Label": labels
})

print("🔍 BiLSTM Predictions on Unseen Text:\n")
display(pred_df)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 352ms/step
🔍 BiLSTM Predictions on Unseen Text:
```


```
{"summary":"{\n  \"name\": \"pred_df\",\n  \"rows\": 7,\n  \"fields\":
[\n    {\n      \"column\": \"Statement\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 7,\n
\"samples\": [\n          \"I can\\u2019t sleep and my thoughts are
racing all night.\",\n          \"Yesterday I cleaned the entire house
in excitement, today I can\\u2019t move.\",\n          \"I don\\u2019t
want to live anymore, I\\u2019m tired of everything.\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Predicted Label\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n
\"Depression\",\n          \"Suicidal\",\n          \"Personality
disorder\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"pred_df"}
```


```
import numpy as np

# Re-sample a small subset of original data
subset_idx = np.random.choice(len(Xtr_pad), size=2000, replace=False)
X_old = Xtr_pad[subset_idx]
y_old = y_train_cat[subset_idx]

# Make sure both old and new have compatible dimensions
min_len = min(len(X_old), len(y_old))
X_old = X_old[:min_len]
y_old = y_old[:min_len]

# Combine old + augmented new data
X_mini = np.concatenate([X_old, pads], axis=0)
y_mini = np.concatenate([y_old, labs], axis=0)

# 🔧 Fix possible mismatch by trimming to equal size
min_len = min(len(X_mini), len(y_mini))
X_mini = X_mini[:min_len]
y_mini = y_mini[:min_len]

print("📏 Final aligned shapes:", X_mini.shape, y_mini.shape)

# 🚀 Fine-tune the model briefly (1 epoch)
bilstm.fit(X_mini, y_mini, epochs=1, batch_size=32, verbose=1)
```

```
 Final aligned shapes: (2007, 100) (2007, 7)
63/63 ━━━━━━━━━━━━━━━━━━━━ 84s 1s/step - accuracy: 0.5453 - loss:
1.1821

<keras.src.callbacks.history.History at 0x7b08d3fdd0d0>

# ------------------------------------------------------------
#  PREDICTION BLOCK — test the trained BiLSTM on new sentences
# ------------------------------------------------------------
import pandas as pd
import numpy as np
from tensorflow.keras.preprocessing.sequence import pad_sequences

#  New unseen sentences to test
new_texts = [
    "I can't sleep and my thoughts are racing all night.",
    "Yesterday I cleaned the entire house in excitement, today I can't
move.",
    "Everything feels dull and meaningless lately.",
    "Too many deadlines are making me anxious and tired.",
    "I feel perfectly fine today, calm and happy.",
    "I don't want to live anymore, I'm tired of everything.",
    "I can't control my emotions and my relationships always fall
apart."
]

#  Convert to padded sequences using your fitted tokenizer
seqs = tokenizer.texts_to_sequences(new_texts)
pads = pad_sequences(seqs, maxlen=max_len, padding='post')

#  Make predictions
pred_probs = bilstm.predict(pads)
pred_classes = np.argmax(pred_probs, axis=1)
labels = le.inverse_transform(pred_classes)

#  Display results in a clean DataFrame
pred_df = pd.DataFrame({
    "Statement": new_texts,
    "Predicted Label": labels
})

print(" BiLSTM Predictions on Unseen Text:\n")
display(pred_df)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 350ms/step
 BiLSTM Predictions on Unseen Text:


{"summary":"{\n  \"name\": \"pred_df\",\n  \"rows\": 7,\n  \"fields\":
[\n    {\n      \"column\": \"Statement\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 7,\n
```

```
\"samples\": [\n          \"I can\\u2019t sleep and my thoughts are
racing all night.\",\n          \"Yesterday I cleaned the entire house
in excitement, today I can\\u2019t move.\",\n          \"I don\\u2019t
want to live anymore, I\\u2019m tired of everything.\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Predicted Label\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n
\"Normal\",\n          \"Depression\",\n          \"Anxiety\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\n}","type":"dataframe","variable_name":"pred_df"}

# 🔁 Targeted refresh for minority classes
final_boost = {
    "Suicidal": [
        "I don’t want to live anymore, I’m tired of everything.",
        "Sometimes I feel death would be easier than this pain.",
        "There’s no reason left for me to keep going."
    ],
    "Bipolar": [
        "I felt unstoppable this morning but hopeless by night.",
        "My energy flips between excitement and exhaustion daily."
    ],
    "Personality disorder": [
        "I trust people one minute and hate them the next.",
        "I love and hate myself at the same time."
    ],
    "Stress": [
        "Deadlines are choking me; I can’t keep up.",
        "Work pressure makes me feel like I’ll collapse."
    ]
}

boost_df = pd.DataFrame([(t,l) for l,L in final_boost.items() for t in
L],
                        columns=["statement","status"])
boost_df["label"] = le.transform(boost_df["status"])

boost_seq = tokenizer.texts_to_sequences(boost_df["statement"])
boost_pad = tf.keras.preprocessing.sequence.pad_sequences(boost_seq,
maxlen=max_len, padding='post')
boost_lab = tf.keras.utils.to_categorical(boost_df["label"],
num_classes=num_classes)

# 🔁 Fine-tune just on these samples for 2 quick epochs
bilstm.fit(boost_pad, boost_lab, epochs=2, batch_size=8, verbose=1)

Epoch 1/2
2/2 ━━━━━━━━━━━━━━━━━━━━ 5s 3s/step - accuracy: 0.0000e+00 - loss:
2.6660
```

```
Epoch 2/2
2/2 ━━━━━━━━━━━━━━━━━━━━ 6s 3s/step - accuracy: 0.0000e+00 - loss:
2.1142

<keras.src.callbacks.history.History at 0x7b08e8b85a00>

# ------------------------------------------------------------
# 🔮 PREDICTION BLOCK — test the trained BiLSTM on new sentences
# ------------------------------------------------------------
import pandas as pd
import numpy as np
from tensorflow.keras.preprocessing.sequence import pad_sequences

# 🧪 New unseen sentences to test
new_texts = [
    "I can’t sleep and my thoughts are racing all night.",
    "Yesterday I cleaned the entire house in excitement, today I can’t
move.",
    "Everything feels dull and meaningless lately.",
    "Too many deadlines are making me anxious and tired.",
    "I feel perfectly fine today, calm and happy.",
    "I don’t want to live anymore, I’m tired of everything.",
    "I can’t control my emotions and my relationships always fall
apart."
]

# 🔄 Convert to padded sequences using your fitted tokenizer
seqs = tokenizer.texts_to_sequences(new_texts)
pads = pad_sequences(seqs, maxlen=max_len, padding='post')

# 🔮 Make predictions
pred_probs = bilstm.predict(pads)
pred_classes = np.argmax(pred_probs, axis=1)
labels = le.inverse_transform(pred_classes)

# 📊 Display results in a clean DataFrame
pred_df = pd.DataFrame({
    "Statement": new_texts,
    "Predicted Label": labels
})

print("🔍 BiLSTM Predictions on Unseen Text:\n")
display(pred_df)

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 419ms/step
🔍 BiLSTM Predictions on Unseen Text:


{"summary":"{\n  \"name\": \"pred_df\",\n  \"rows\": 7,\n  \"fields\":
[\n    {\n      \"column\": \"Statement\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 7,\n
```

```
\"samples\": [\n          \"I can\\u2019t sleep and my thoughts are
racing all night.\",\n          \"Yesterday I cleaned the entire house
in excitement, today I can\\u2019t move.\",\n          \"I don\\u2019t
want to live anymore, I\\u2019m tired of everything.\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Predicted Label\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n
\"Normal\",\n          \"Anxiety\",\n          \"Suicidal\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"pred_df"}

# ------------------------------------------------------------
# 🧠 BiLSTM Prediction Block (with Confidence %)
# ------------------------------------------------------------
import pandas as pd
import numpy as np
from tensorflow.keras.preprocessing.sequence import pad_sequences

# 🔹 New unseen sentences to test
new_texts = [
    "I can’t sleep and my thoughts are racing all night.",
    "Yesterday I cleaned the entire house in excitement, today I can’t
move.",
    "Everything feels dull and meaningless lately.",
    "Too many deadlines are making me anxious and tired.",
    "I feel perfectly fine today, calm and happy.",
    "I don’t want to live anymore, I’m tired of everything.",
    "I can’t control my emotions and my relationships always fall
apart."
]

# 🔹 Tokenize and pad
seqs = tokenizer.texts_to_sequences(new_texts)
pads = pad_sequences(seqs, maxlen=max_len, padding='post')

# 🔹 Predict
pred_probs = bilstm.predict(pads)
pred_classes = np.argmax(pred_probs, axis=1)
labels = le.inverse_transform(pred_classes)
confidences = (pred_probs.max(axis=1) * 100).round(2)

# 🔹 Display with confidence %
pred_df = pd.DataFrame({
    "Statement": new_texts,
    "Predicted Label": labels,
    "Confidence (%)": confidences
})
```

```
print("🔍 BiLSTM Predictions on Unseen Text (with Confidence):\n")
display(pred_df)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 405ms/step
🔍 BiLSTM Predictions on Unseen Text (with Confidence):
```


```
{"summary":"{\n  \"name\": \"pred_df\",\n  \"rows\": 7,\n  \"fields\":
[\n    {\n      \"column\": \"Statement\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 7,\n
\"samples\": [\n          \"I can\\u2019t sleep and my thoughts are
racing all night.\",\n          \"Yesterday I cleaned the entire house
in excitement, today I can\\u2019t move.\",\n          \"I don\\u2019t
want to live anymore, I\\u2019m tired of everything.\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Predicted Label\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n
\"Normal\",\n          \"Anxiety\",\n          \"Suicidal\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Confidence (%)\",\n      \"properties\": {\n        \"dtype\":
\"float32\",\n        \"num_unique_values\": 7,\n        \"samples\":
[\n          37.060001373291016,\n          68.25,\n
54.380001068115234\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"pred_df"}
```


```
# ⚖ Focused data enrichment for minority classes
extra_minor = {
    "Bipolar": [
        "I felt unstoppable this morning and completely hopeless by
evening.",
        "My energy swings between extreme joy and exhaustion.",
        "I start so many things full of excitement and abandon them
suddenly.",
        "One day I love life, the next I can’t get out of bed."
    ],
    "Personality disorder": [
        "I push people away even when I want them close.",
        "My relationships collapse because I can’t control my
emotions.",
        "I love someone deeply one moment and hate them the next.",
        "I can’t maintain steady relationships because my mood changes
fast."
    ],
    "Stress": [
        "Work pressure is suffocating; I can’t sleep or focus.",
        "Deadlines are crushing me; I’m always tense and tired.",
        "My head hurts from overthinking about tasks every day.",
```

```
        "I feel like I’m carrying the world on my shoulders."
    ]
}

aug_df2 = pd.DataFrame([(t, l) for l, L in extra_minor.items() for t
in L],
                       columns=["statement", "status"])
aug_df2["label"] = le.transform(aug_df2["status"])

# Tokenize + pad
seqs = tokenizer.texts_to_sequences(aug_df2["statement"])
pads = tf.keras.preprocessing.sequence.pad_sequences(seqs,
maxlen=max_len, padding='post')
labs = tf.keras.utils.to_categorical(aug_df2["label"],
num_classes=num_classes)

# Fine-tune briefly (light gradient refresh)
bilstm.fit(pads, labs, epochs=2, batch_size=8, verbose=1)
```

```
Epoch 1/2
2/2 ━━━━━━━━━━━━━━━━━━━━ 5s 2s/step - accuracy: 0.0556 - loss: 2.3415
Epoch 2/2
2/2 ━━━━━━━━━━━━━━━━━━━━ 6s 2s/step - accuracy: 0.1528 - loss: 1.8984

<keras.src.callbacks.history.History at 0x7b090cb84b90>
```

```
subset_idx = np.random.choice(len(Xtr_pad), size=1000, replace=False)
X_mix = np.concatenate([Xtr_pad[subset_idx], pads], axis=0)
y_mix = np.concatenate([y_train_cat[subset_idx], labs], axis=0)

min_len = min(len(X_mix), len(y_mix))
X_mix, y_mix = X_mix[:min_len], y_mix[:min_len]

# Short stabilization pass
bilstm.fit(X_mix, y_mix, epochs=1, batch_size=32, verbose=1)
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 43s 1s/step - accuracy: 0.7210 - loss:
0.7968

<keras.src.callbacks.history.History at 0x7b08d3fde450>
```

```
new_texts = [
    "I can’t sleep and my thoughts are racing all night.",
    "Yesterday I cleaned the entire house in excitement, today I can’t
move.",
    "Everything feels dull and meaningless lately.",
    "Too many deadlines are making me anxious and tired.",
    "I feel perfectly fine today, calm and happy.",
    "I don’t want to live anymore, I’m tired of everything.",
    "I can’t control my emotions and my relationships always fall
apart.",
```

```
    "My workload is overwhelming, I feel constant pressure to
perform.",
    "I love people one day and hate them the next.",
    "My mood flips from excitement to despair in hours."
]

seqs = tokenizer.texts_to_sequences(new_texts)
pads = tf.keras.preprocessing.sequence.pad_sequences(seqs,
maxlen=max_len, padding='post')
pred_probs = bilstm.predict(pads)
pred_classes = np.argmax(pred_probs, axis=1)
labels = le.inverse_transform(pred_classes)
conf = (pred_probs.max(axis=1) * 100).round(2)

pd.DataFrame({"Statement": new_texts, "Predicted Label": labels,
"Confidence %": conf})
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 368ms/step
```


```
{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 10,\n  \"fields\": [\n
{\n      \"column\": \"Statement\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 10,\n
\"samples\": [\n          \"I love people one day and hate them the
next.\",\n          \"Yesterday I cleaned the entire house in
excitement, today I can\\u2019t move.\",\n          \"I don\\u2019t
want to live anymore, I\\u2019m tired of everything.\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Predicted Label\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 5,\n        \"samples\": [\n
\"Bipolar\",\n          \"Personality disorder\",\n
\"Anxiety\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Confidence %\",\n      \"properties\": {\n        \"dtype\":
\"float32\",\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          80.41999816894531,\n          89.0,\n
21.81999969482422\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
stress_boost = {
    "Stress": [
        "I’m under too much pressure, I can’t take it anymore.",
        "Deadlines are killing me; I’m exhausted and tense.",
        "My heart races every time I see my work piled up."
    ]
}

import pandas as pd, numpy as np, tensorflow as tf

boost_df = pd.DataFrame([(t,l) for l,L in stress_boost.items() for t
```

```
in L],
                          columns=["statement","status"])
boost_df["label"] = le.transform(boost_df["status"])

seqs = tokenizer.texts_to_sequences(boost_df["statement"])
pads = tf.keras.preprocessing.sequence.pad_sequences(seqs,
maxlen=max_len, padding='post')
labs = tf.keras.utils.to_categorical(boost_df["label"],
num_classes=num_classes)

bilstm.fit(pads, labs, epochs=1, batch_size=8, verbose=1)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step - accuracy: 0.3333 - loss: 3.3548
```

```
<keras.src.callbacks.history.History at 0x7b08d3ea4ec0>
```

```
new_texts = [
    "I can’t sleep and my thoughts are racing all night.",
    "Yesterday I cleaned the entire house in excitement, today I can’t
move.",
    "Everything feels dull and meaningless lately.",
    "Too many deadlines are making me anxious and tired.",
    "I feel perfectly fine today, calm and happy.",
    "I don’t want to live anymore, I’m tired of everything.",
    "I can’t control my emotions and my relationships always fall
apart.",
    "My workload is overwhelming, I feel constant pressure to
perform.",
    "I love people one day and hate them the next.",
    "My mood flips from excitement to despair in hours."
]

seqs = tokenizer.texts_to_sequences(new_texts)
pads = tf.keras.preprocessing.sequence.pad_sequences(seqs,
maxlen=max_len, padding='post')
pred_probs = bilstm.predict(pads)
pred_classes = np.argmax(pred_probs, axis=1)
labels = le.inverse_transform(pred_classes)
conf = (pred_probs.max(axis=1) * 100).round(2)

pd.DataFrame({"Statement": new_texts, "Predicted Label": labels,
"Confidence %": conf})

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Bidirectional,
Dense, Dropout
import numpy as np, pandas as pd

# same parameters as before
embed_dim = 100
```

```
max_len = 100
num_classes = 7     # your 7 labels
num_words = 20000   # or whatever vocab_size you used

def build_bilstm(num_words, embed_dim, max_len, num_classes):
    model = Sequential([
        Embedding(num_words, embed_dim, input_length=max_len),
        Bidirectional(LSTM(128, dropout=0.3, recurrent_dropout=0.3,
return_sequences=True)),
        Bidirectional(LSTM(64, dropout=0.3, recurrent_dropout=0.3)),
        Dense(128, activation='relu'),
        Dropout(0.3),
        Dense(num_classes, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
    return model

bilstm = build_bilstm(num_words, embed_dim, max_len, num_classes)
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/
embedding.py:97: UserWarning: Argument `input_length` is deprecated.
Just remove it.
  warnings.warn(
```

```
from tensorflow.keras.models import load_model

# directly load the full trained model
bilstm = load_model("bilstm_best.h5")

print("□ Model loaded successfully (architecture + weights)!")
bilstm.summary()
```

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

□ Model loaded successfully (architecture + weights)!

Model: "sequential_3"
```

```
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━
┓
┃ Layer (type)                         ┃ Output Shape                ┃
Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━
┩
│ embedding_3 (Embedding)              │ (None, 100, 200)            │
4,000,000 │
├──────────────────────────────────────┼─────────────────────────────┼─────────────────
┤
```

```
| bidirectional (Bidirectional)   | (None, 256)             |
336,896 |
├─────────────────────────────────┼─────────────────────────┼──────────
──────┤
| dropout_3 (Dropout)             | (None, 256)             |
0 |
├─────────────────────────────────┼─────────────────────────┼──────────
──────┤
| dense_3 (Dense)                 | (None, 7)               |
1,799 |
└─────────────────────────────────┴─────────────────────────┴──────────
──────┘

 Total params: 4,338,697 (16.55 MB)

 Trainable params: 4,338,695 (16.55 MB)

 Non-trainable params: 0 (0.00 B)

 Optimizer params: 2 (12.00 B)
```

```python
from tensorflow.keras.preprocessing.text import Tokenizer
from sklearn.preprocessing import LabelEncoder
import pandas as pd

# 📂 Load your dataset again (update path if needed)
df = pd.read_csv(
    "/content/drive/MyDrive/NLP PROJECT/Combined Data (1).csv",
    encoding="ISO-8859-1",
    on_bad_lines="skip",    # skips rows with too many/few columns
    engine="python"         # more forgiving parser
)

df.dropna(subset=["statement", "status"], inplace=True)

# 🔤 Tokenizer (same settings as before)
vocab_size = 20000
tokenizer = Tokenizer(num_words=vocab_size, oov_token="<OOV>")
tokenizer.fit_on_texts(df["statement"])

# 🏷 Label Encoder (for decoding predictions)
le = LabelEncoder()
le.fit(df["status"])

print("✅ Tokenizer & LabelEncoder restored successfully!")
print("Classes:", list(le.classes_))
```

```
✅ Tokenizer & LabelEncoder restored successfully!
Classes: [' I stopped hesitating to talk to handsome guys=9', ' an
hour on the bike \x13 has a magical ability to burn off that anxiety
and re-set all those neurotransmitters. It will also help you get that
```

```
good night\x19s sleep.', ' deal with the important issues over the
urgent issues: here-in lies the path to living sustainably crisis
free.', 'Anxiety', 'Bipolar', 'Depression', 'Normal', 'Personality
disorder', 'Stress', 'Suicidal']
```

```
# Keep only the 7 valid emotional categories
valid_labels = [
    "Anxiety",
    "Bipolar",
    "Depression",
    "Normal",
    "Personality disorder",
    "Stress",
    "Suicidal"
]

# Filter and rebuild encoder on only these
df = df[df["status"].isin(valid_labels)].copy()

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
le.fit(valid_labels)

print("□ Cleaned label encoder classes:")
print(le.classes_)
```

```
□ Cleaned label encoder classes:
['Anxiety' 'Bipolar' 'Depression' 'Normal' 'Personality disorder'
'Stress'
 'Suicidal']
```

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np, pandas as pd

new_texts = [
    "I can’t sleep and my thoughts are racing all night.",
    "Yesterday I cleaned the entire house in excitement, today I can’t
move.",
    "Everything feels dull and meaningless lately.",
    "Too many deadlines are making me anxious and tired.",
    "I feel perfectly fine today, calm and happy.",
    "I don’t want to live anymore, I’m tired of everything.",
    "I can’t control my emotions and my relationships always fall
apart.",
    "My workload is overwhelming, I feel constant pressure to
perform.",
    "I love people one day and hate them the next.",
    "My mood flips from excitement to despair in hours."
]
```

```
seqs = tokenizer.texts_to_sequences(new_texts)
pads = pad_sequences(seqs, maxlen=100, padding='post')

pred_probs = bilstm.predict(pads)
pred_classes = np.argmax(pred_probs, axis=1)
labels = le.inverse_transform(pred_classes)
conf = (pred_probs.max(axis=1) * 100).round(2)

pd.DataFrame({"Statement": new_texts, "Predicted Label": labels,
"Confidence %": conf})
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
```


```
{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 10,\n  \"fields\": [\n
{\n      \"column\": \"Statement\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 10,\n
\"samples\": [\n          \"I love people one day and hate them the
next.\",\n          \"Yesterday I cleaned the entire house in
excitement, today I can\\u2019t move.\",\n          \"I don\\u2019t
want to live anymore, I\\u2019m tired of everything.\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Predicted Label\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n
\"Normal\",\n          \"Suicidal\",\n          \"Personality
disorder\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Confidence %\",\n      \"properties\": {\n        \"dtype\":
\"float32\",\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          74.3499984741211,\n          88.43000030517578,\n
72.4800033569336\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
from sklearn.preprocessing import LabelEncoder

# Manually set the correct order (this was the order during training)
label_order = [
    "Anxiety",
    "Bipolar",
    "Depression",
    "Normal",
    "Personality disorder",
    "Stress",
    "Suicidal"
]

le = LabelEncoder()
le.classes_ = np.array(label_order)
```

```
print("□ Fixed label mapping:")
print(le.classes_)
```

```
□ Fixed label mapping:
['Anxiety' 'Bipolar' 'Depression' 'Normal' 'Personality disorder'
'Stress'
 'Suicidal']
```

```
pred_probs = bilstm.predict(pads)
pred_classes = np.argmax(pred_probs, axis=1)
labels = le.inverse_transform(pred_classes)
conf = (pred_probs.max(axis=1) * 100).round(2)

pd.DataFrame({
    "Statement": new_texts,
    "Predicted Label": labels,
    "Confidence %": conf
})
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 263ms/step
```


```
{"summary":"{\n  \"name\": \"})\",\n  \"rows\": 10,\n  \"fields\": [\n
{\n      \"column\": \"Statement\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 10,\n
\"samples\": [\n          \"I love people one day and hate them the
next.\",\n          \"Yesterday I cleaned the entire house in
excitement, today I can\\u2019t move.\",\n          \"I don\\u2019t
want to live anymore, I\\u2019m tired of everything.\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Predicted Label\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 3,\n        \"samples\": [\n
\"Normal\",\n          \"Suicidal\",\n          \"Personality
disorder\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Confidence %\",\n      \"properties\": {\n        \"dtype\":
\"float32\",\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          74.3499984741211,\n          88.43000030517578,\n
72.4800033569336\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
import numpy as np, pandas as pd
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.preprocessing import LabelEncoder

# 1) Ensure df is clean and only has the 7 valid classes
valid_labels = ["Anxiety","Bipolar","Depression","Normal","Personality
disorder","Stress","Suicidal"]
df = df[df["status"].isin(valid_labels)].copy()

# Recreate a clean encoder (order here is just for y_true indexing;
```

```
we'll learn the model order)
le_true = LabelEncoder()
le_true.fit(valid_labels)  # [Anxiety, Bipolar, Depression, Normal,
Personality disorder, Stress, Suicidal]

# 2) Take a manageable subset to infer mapping fast
N = min(5000, len(df))
sample = df.sample(N, random_state=42).reset_index(drop=True)

X_seq = tokenizer.texts_to_sequences(sample["statement"])
X_pad = pad_sequences(X_seq, maxlen=100, padding='post')
y_true_idx = le_true.transform(sample["status"])

# 3) Get raw model predictions (indices 0..6 in the model's internal
order)
probs = bilstm.predict(X_pad, verbose=0)
y_pred_idx = probs.argmax(axis=1)

# 4) Build contingency matrix: rows=true (7), cols=pred_index (7)
K = 7
mat = np.zeros((K, K), dtype=int)
for t, p in zip(y_true_idx, y_pred_idx):
    mat[t, p] += 1

# 5) Solve assignment: map each predicted index -> best true label
#    We want to maximize correct counts, so minimize negative counts.
try:
    from scipy.optimize import linear_sum_assignment
    cost = -mat  # maximize mat by minimizing negative
    r_ind, c_ind = linear_sum_assignment(cost)
    # r_ind: true label idx; c_ind: pred index; map pred -> true
    pred_to_true = {}
    for r, c in zip(r_ind, c_ind):
        pred_to_true[c] = r
except Exception:
    # Fallback greedy (one-to-one best matches)
    pred_to_true = {}
    used_true = set()
    for pred_col in range(K):
        # pick the true row with max count not taken
        row_order = np.argsort(-mat[:, pred_col])
        for r in row_order:
            if r not in used_true:
                pred_to_true[pred_col] = r
                used_true.add(r)
                break

# 6) Build readable mapping: model index -> class name
model_index_to_label = {pred_idx: le_true.classes_[true_idx] for
pred_idx, true_idx in pred_to_true.items()}
```

```
print("🔍 Learned mapping (model index → label):")
for i in range(K):
    print(f"  {i} → {model_index_to_label.get(i, 'UNKNOWN')}")

# 7) Define a helper to predict with the corrected mapping
def predict_with_mapping(texts):
    seqs = tokenizer.texts_to_sequences(texts)
    pads = pad_sequences(seqs, maxlen=100, padding='post')
    pr = bilstm.predict(pads, verbose=0)
    raw_idx = pr.argmax(axis=1)
    mapped_labels = [model_index_to_label[i] for i in raw_idx]
    conf = (pr.max(axis=1) * 100).round(2)
    return pd.DataFrame({"Statement": texts, "Predicted Label":
mapped_labels, "Confidence %": conf})

# 8) Re-run on your unseen examples
new_texts = [
    "I can’t sleep and my thoughts are racing all night.",
    "Yesterday I cleaned the entire house in excitement, today I can’t
move.",
    "Everything feels dull and meaningless lately.",
    "Too many deadlines are making me anxious and tired.",
    "I feel perfectly fine today, calm and happy.",
    "I don’t want to live anymore, I’m tired of everything.",
    "I can’t control my emotions and my relationships always fall
apart.",
    "My workload is overwhelming, I feel constant pressure to
perform.",
    "I love people one day and hate them the next.",
    "My mood flips from excitement to despair in hours."
]

fixed_preds = predict_with_mapping(new_texts)
fixed_preds
```

```
🔍 Learned mapping (model index → label):
  0 → Bipolar
  1 → Stress
  2 → Depression
  3 → Normal
  4 → Personality disorder
  5 → Anxiety
  6 → Suicidal
```


```
{"summary":"{\n  \"name\": \"fixed_preds\",\n  \"rows\": 10,\n
\"fields\": [\n    {\n      \"column\": \"Statement\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n        \"samples\": [\n          \"I love
people one day and hate them the next.\",\n          \"Yesterday I
```

```
cleaned the entire house in excitement, today I can\\u2019t move.\",\n
\"I don\\u2019t want to live anymore, I\\u2019m tired of
everything.\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Predicted Label\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"Normal\",\n          \"Suicidal\",\n
\"Personality disorder\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Confidence %\",\n      \"properties\": {\n
\"dtype\": \"float32\",\n        \"num_unique_values\": 10,\n
\"samples\": [\n          74.3499984741211,\n
88.43000030517578,\n          72.4800033569336\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"fixed_preds"}
```


```
pd.DataFrame(pred_probs, columns=[model_index_to_label[i] for i in
range(7)])
```


```
{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 10,\n  \"fields\": [\n
{\n      \"column\": \"Bipolar\",\n      \"properties\": {\n
\"dtype\": \"float32\",\n        \"num_unique_values\": 10,\n
\"samples\": [\n          0.0009304233826696873,\n
0.0010933664161711931,\n          0.01918850466609001\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Stress\",\n      \"properties\":
{\n        \"dtype\": \"float32\",\n        \"num_unique_values\":
10,\n        \"samples\": [\n          0.001205894281156361,\n
0.0019090473651885986,\n          0.006055960897356272\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Depression\",\n
\"properties\": {\n        \"dtype\": \"float32\",\n
\"num_unique_values\": 10,\n        \"samples\": [\n
0.25026440620422363,\n          0.10127780586481094,\n
0.09724167734384537\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Normal\",\n      \"properties\": {\n        \"dtype\": \"float32\",\
n        \"num_unique_values\": 10,\n        \"samples\": [\n
0.0026036598719656467,\n          0.004216973669826984,\n
0.7248243689537048\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Personality disorder\",\n      \"properties\": {\n        \"dtype\":
\"float32\",\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          0.00025706563610583544,\n
0.00015652261208742857,\n          0.0014367044204846025\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Anxiety\",\n      \"properties\":
{\n        \"dtype\": \"float32\",\n        \"num_unique_values\":
10,\n        \"samples\": [\n          0.0012862264411523938,\n
0.0070262993685901165,\n          0.01957395300269127\n        ],\n
```

\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Suicidal\",\n      \"properties\": {\n        \"dtype\": \"float32\",\n        \"num_unique_values\": 10,\n        \"samples\": [\n          0.7434523105621338,\n          0.8843199610710144,\n          0.13167887926101685\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

```
# ------------------------------------------------------------
# 🔹 1. Create a small balanced dataset (1–2 examples per class)
# ------------------------------------------------------------
minor_boost = {
    "Anxiety": [
        "My heart races every time I think about tomorrow.",
        "I worry constantly about small things I can’t control."
    ],
    "Bipolar": [
        "I felt on top of the world this morning and worthless by night.",
        "My energy flips between excitement and exhaustion every day."
    ],
    "Depression": [
        "Everything feels heavy and meaningless lately.",
        "I have no motivation to do anything anymore."
    ],
    "Normal": [
        "I feel calm and content today.",
        "It’s been a peaceful day and I’m relaxed."
    ],
    "Personality disorder": [
        "I love someone deeply one moment and push them away the next.",
        "My relationships break down because my emotions change so fast."
    ],
    "Stress": [
        "Deadlines are suffocating me; I can’t keep up.",
        "My workload feels overwhelming and I can’t relax."
    ],
    "Suicidal": [
        "I don’t want to live anymore; everything feels pointless.",
        "Sometimes I wish I could disappear forever."
    ]
}

import pandas as pd, numpy as np, tensorflow as tf
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.optimizers import Adam

# ------------------------------------------------------------
```

```
# 2️⃣  Build DataFrame and map correct label indices
# ------------------------------------------------------------
boost_df = pd.DataFrame([(t, l) for l, L in minor_boost.items() for t
in L],
                        columns=["statement", "status"])

# Reverse the dictionary (label → model_index)
label_to_model_index = {v: k for k, v in model_index_to_label.items()}

# Assign numeric label based on your model's internal order
boost_df["label"] = [label_to_model_index[lbl] for lbl in
boost_df["status"]]

# ------------------------------------------------------------
# 3️⃣  Tokenize + pad + one-hot encode
# ------------------------------------------------------------
boost_seq = tokenizer.texts_to_sequences(boost_df["statement"])
boost_pad = pad_sequences(boost_seq, maxlen=100, padding='post')
boost_lab = tf.keras.utils.to_categorical(boost_df["label"],
num_classes=7)

# ------------------------------------------------------------
# 4️⃣  Fine-tune lightly (very small LR to preserve prior learning)
# ------------------------------------------------------------
bilstm.compile(optimizer=Adam(learning_rate=1e-5),
loss='categorical_crossentropy', metrics=['accuracy'])
bilstm.fit(boost_pad, boost_lab, epochs=2, batch_size=8, verbose=1)

print("✅ Mini fine-tune complete — model reinforced for all 7
classes!")
```

```
Epoch 1/2
2/2 ━━━━━━━━━━━━━━━━━━━━ 32s 1s/step - accuracy: 0.1369 - loss: 3.9784
Epoch 2/2
2/2 ━━━━━━━━━━━━━━━━━━━━ 3s 1s/step - accuracy: 0.0893 - loss: 4.1842
✅ Mini fine-tune complete — model reinforced for all 7 classes!
```

```
fixed_preds = predict_with_mapping(new_texts)
display(fixed_preds)
```


```
{"summary":"{\n  \"name\": \"fixed_preds\",\n  \"rows\": 10,\n
\"fields\": [\n    {\n      \"column\": \"Statement\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 10,\n        \"samples\": [\n          \"I love
people one day and hate them the next.\",\n          \"Yesterday I
cleaned the entire house in excitement, today I can\\u2019t move.\",\n
\"I don\\u2019t want to live anymore, I\\u2019m tired of
everything.\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Predicted Label\",\n      \"properties\": {\n        \"dtype\":
```

```
\"category\",\n        \"num_unique_values\": 3,\n        \"samples\":
[\n          \"Normal\",\n          \"Suicidal\",\n
\"Personality disorder\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Confidence %\",\n      \"properties\": {\n
\"dtype\": \"float32\",\n        \"num_unique_values\": 10,\n
\"samples\": [\n          74.30999755859375,\n
88.4000015258789,\n          72.55000305175781\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"fixed_preds"}

import tensorflow as tf
import pandas as pd, numpy as np
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import load_model
from sklearn.preprocessing import LabelEncoder

# --- 1 Load model ---
bilstm = load_model("bilstm_best.h5")

# --- 2 Reload and clean dataset ---
df = pd.read_csv("/content/drive/MyDrive/NLP PROJECT/Combined Data
(1).csv",
                 encoding="ISO-8859-1", on_bad_lines="skip",
engine="python")
valid_labels = ["Anxiety","Bipolar","Depression","Normal",
                "Personality disorder","Stress","Suicidal"]
df = df[df["status"].isin(valid_labels)].copy()

# --- 3 Tokenizer & encoder ---
tokenizer = Tokenizer(num_words=20000, oov_token="<OOV>")

# 🧹 Clean text column to avoid float/NaN errors
df["statement"] = df["statement"].astype(str)
df = df[df["statement"].str.strip() != ""]   # remove empty rows

tokenizer.fit_on_texts(df["statement"])

# 🏷 Label encoder
le = LabelEncoder()
le.fit(valid_labels)


# --- 4 The correct internal label order from training ---
model_index_to_label = {
    0: "Bipolar",
    1: "Stress",
    2: "Depression",
    3: "Normal",
```

```
    4: "Personality disorder",
    5: "Anxiety",
    6: "Suicidal"
}
label_to_model_index = {v:k for k,v in model_index_to_label.items()}
```

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

```
# Balanced micro-dataset (2 samples per class)
mini = {
    "Anxiety": ["I'm always on edge and can't stop worrying.",
                "Even small noises make me nervous these days."],
    "Bipolar": ["I felt unstoppable this morning and hopeless by
night.",
                "My mood flips from extreme energy to deep sadness."],
    "Depression": ["Nothing interests me anymore; life feels heavy.",
                   "I'm tired all the time and feel empty inside."],
    "Normal": ["I feel calm and peaceful today.",
               "Everything is fine; I'm relaxed and happy."],
    "Personality disorder": ["I trust people instantly then push them
away.",
                             "My emotions change too quickly to
control."],
    "Stress": ["Deadlines are suffocating me; I can't breathe.",
               "Too many tasks make me feel overwhelmed."],
    "Suicidal": ["I don't want to live anymore.",
                 "Sometimes I wish I could just disappear."]
}

boost_df = pd.DataFrame([(t,l) for l,L in mini.items() for t in L],
                        columns=["text","label"])
boost_df["label_id"] = [label_to_model_index[l] for l in
boost_df["label"]]

seqs = tokenizer.texts_to_sequences(boost_df["text"])
pads = pad_sequences(seqs, maxlen=100, padding='post')
labs = tf.keras.utils.to_categorical(boost_df["label_id"],
num_classes=7)

# fine-tune lightly
from tensorflow.keras.optimizers import Adam
bilstm.compile(optimizer=Adam(learning_rate=1e-5),
               loss='categorical_crossentropy', metrics=['accuracy'])
bilstm.fit(pads, labs, epochs=2, batch_size=8, verbose=1)
```

```
Epoch 1/2
2/2 ━━━━━━━━━━━━━━━━━━━━ 20s 723ms/step - accuracy: 0.1845 - loss:
3.5535
```

```
Epoch 2/2
2/2 ━━━━━━━━━━━━━━━━━━━━ 2s 836ms/step - accuracy: 0.1369 - loss:
3.5931

<keras.src.callbacks.history.History at 0x7f1e92e71430>
```

```
test_texts = [
    "I can’t sleep and my thoughts are racing all night.",
    "Yesterday I cleaned the entire house in excitement, today I can’t
move.",
    "Everything feels dull and meaningless lately.",
    "Too many deadlines are making me anxious and tired.",
    "I feel perfectly fine today, calm and happy.",
    "I don’t want to live anymore, I’m tired of everything.",
    "I can’t control my emotions and my relationships always fall
apart.",
    "My workload is overwhelming, I feel constant pressure to
perform.",
    "I love people one day and hate them the next.",
    "My mood flips from excitement to despair in hours."
]

X = pad_sequences(tokenizer.texts_to_sequences(test_texts),
maxlen=100, padding='post')
pred_probs = bilstm.predict(X)
pred_idx = pred_probs.argmax(axis=1)
pred_labels = [model_index_to_label[i] for i in pred_idx]
conf = (pred_probs.max(axis=1) * 100).round(2)

pd.DataFrame({"Statement": test_texts,
              "Predicted Label": pred_labels,
              "Confidence %": conf})
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step
```


```
{"summary":"{\n  \"name\": \"              \\\"Confidence %\\\":
conf})\",\n  \"rows\": 10,\n  \"fields\": [\n    {\n      \"column\":
\"Statement\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          \"I love people one day and hate them the next.\",\n
\"Yesterday I cleaned the entire house in excitement, today I can\\
u2019t move.\",\n          \"I don\\u2019t want to live anymore, I\\
u2019m tired of everything.\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Predicted Label\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 4,\n
\"samples\": [\n          \"Suicidal\",\n          \"Normal\",\n
\"Bipolar\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Confidence %\",\n      \"properties\": {\n        \"dtype\":
```

```
\"float32\",\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          74.29000091552734,\n          85.61000061035156,\n
72.0\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

# Extra disambiguation samples for confusing classes
extra = {
    "Normal": [
        "I feel relaxed and in control of my day.",
        "Everything is stable; nothing is bothering me right now."
    ],
    "Stress": [
        "My workload keeps piling up and I feel constant pressure.",
        "I have too many tasks and deadlines; I can't unwind."
    ],
    "Suicidal": [
        "I see no reason to keep living; everything feels hopeless.",
        "I want to end my pain and disappear forever."
    ],
    "Personality disorder": [
        "My emotions switch instantly; I love someone one minute and
hate them the next.",
        "People tell me I’m unpredictable and too intense in
relationships."
    ]
}

# Build quick DataFrame
extra_df = pd.DataFrame([(t, l) for l,L in extra.items() for t in L],
                        columns=["text","label"])
extra_df["label_id"] = [label_to_model_index[l] for l in
extra_df["label"]]

# Tokenize + pad + one-hot encode
seqs = tokenizer.texts_to_sequences(extra_df["text"])
pads = pad_sequences(seqs, maxlen=100, padding='post')
labs = tf.keras.utils.to_categorical(extra_df["label_id"],
num_classes=7)

# Light fine-tune
bilstm.compile(optimizer=tf.keras.optimizers.Adam(1e-5),
               loss='categorical_crossentropy', metrics=['accuracy'])
bilstm.fit(pads, labs, epochs=1, batch_size=8, verbose=1)

print("□ Boundary fine-tune complete —
stress/normal/suicidal/personality clarified.")

1/1 ━━━━━━━━━━━━━━━━━━━━ 13s 13s/step - accuracy: 0.3750 - loss:
3.4101
```

```
 Boundary fine-tune complete — stress/normal/suicidal/personality clarified.

fixed_preds = predict_with_mapping(new_texts)
display(fixed_preds)
```


```
{"summary":"{\n  \"name\": \"fixed_preds\",\n  \"rows\": 10,\n  \"fields\": [\n    {\n      \"column\": \"Statement\",\n      \"properties\": {\n        \"dtype\": \"string\",\n        \"num_unique_values\": 10,\n        \"samples\": [\n          \"I love people one day and hate them the next.\",\n          \"Yesterday I cleaned the entire house in excitement, today I can\\u2019t move.\",\n          \"I don\\u2019t want to live anymore, I\\u2019m tired of everything.\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Predicted Label\",\n      \"properties\": {\n        \"dtype\": \"category\",\n        \"num_unique_values\": 4,\n        \"samples\": [\n          \"Suicidal\",\n          \"Normal\",\n          \"Bipolar\"\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    },\n    {\n      \"column\": \"Confidence %\",\n      \"properties\": {\n        \"dtype\": \"float32\",\n        \"num_unique_values\": 9,\n        \"samples\": [\n          74.30000305175781,\n          77.66000366210938,\n          78.94000244140625\n        ],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe","variable_name":"fixed_preds"}
```


```
!pip install -q transformers datasets evaluate accelerate

import pandas as pd, numpy as np, torch
from datasets import Dataset
from transformers import AutoTokenizer, AutoModelForSequenceClassification, TrainingArguments, Trainer
import evaluate
from sklearn.model_selection import train_test_split

# Adjust the path if yours is different
df = pd.read_csv(
    "/content/drive/MyDrive/NLP PROJECT/Combined Data (1).csv",
    encoding="ISO-8859-1",
    on_bad_lines="skip",
    engine="python"
)

valid_labels = ["Anxiety","Bipolar","Depression","Normal","Personality disorder","Stress","Suicidal"]

# Keep only valid rows and ensure 'statement' is proper text
df = df[df["status"].isin(valid_labels)].copy()
```

```
df["statement"] = df["statement"].astype(str)
df = df[df["statement"].str.strip() != ""]

# Cap to at most 600 per class for speed; raise to 1000 if CPU is okay
df_bal = df.groupby("status").apply(
    lambda x: x.sample(n=min(len(x), 600), random_state=42)
).reset_index(drop=True)

# Stratified split with sklearn (works with string labels)
train_df, test_df = train_test_split(
    df_bal, test_size=0.2, stratify=df_bal["status"], random_state=42
)

# Convert to HuggingFace datasets with expected column names
train_ds =
Dataset.from_pandas(train_df.rename(columns={"statement":"text","statu
s":"labels"}))
test_ds  =
Dataset.from_pandas(test_df.rename(columns={"statement":"text","status
":"labels"}))
```

```
/tmp/ipython-input-2657938110.py:2: DeprecationWarning:
DataFrameGroupBy.apply operated on the grouping columns. This behavior
is deprecated, and in a future version of pandas the grouping columns
will be excluded from the operation. Either pass
`include_groups=False` to exclude the groupings or explicitly select
the grouping columns after groupby to silence this warning.
  df_bal = df.groupby("status").apply(
```

```
tok = AutoTokenizer.from_pretrained("distilbert-base-uncased")

def tokenize(batch):
    return tok(batch["text"], truncation=True, padding="max_length",
max_length=128)

label2id = {l:i for i,l in enumerate(valid_labels)}
id2label = {i:l for l,i in label2id.items()}

# Tokenize and convert string labels → numeric ids
train_tok = train_ds.map(tokenize, batched=True)
test_tok  = test_ds.map(tokenize, batched=True)

train_tok = train_tok.map(lambda e: {"label":[label2id[x] for x in
e["labels"]]}, batched=True)
test_tok  = test_tok.map(lambda e: {"label":[label2id[x] for x in
e["labels"]]}, batched=True)

train_tok = train_tok.remove_columns(["text","labels"])
test_tok  = test_tok.remove_columns(["text","labels"])
```

```
train_tok.set_format("torch")
test_tok.set_format("torch")
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/
_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your
settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to
access public models or datasets.
  warnings.warn(
```


```
{"model_id":"d374ae15f5c44d619b6ce662f08410d1","version_major":2,"vers
ion_minor":0}

{"model_id":"aa97e39430c84459b9db4eb9c44c4d99","version_major":2,"vers
ion_minor":0}

{"model_id":"68a00b2028214dc09fc8ddb83faaecdf","version_major":2,"vers
ion_minor":0}

{"model_id":"b74089761baa4f2fb7c8b28b696c04ab","version_major":2,"vers
ion_minor":0}

{"model_id":"99d60cdfc5d9498fbe0556afe7054ac6","version_major":2,"vers
ion_minor":0}

{"model_id":"ac99e65763ab4f9db32f544c114106ce","version_major":2,"vers
ion_minor":0}

{"model_id":"b279e6b9a4f54393895d1a4386f0404e","version_major":2,"vers
ion_minor":0}

{"model_id":"3d5c53d277ed41f0b7a41015294e35bd","version_major":2,"vers
ion_minor":0}
```


```
model = AutoModelForSequenceClassification.from_pretrained(
    "distilbert-base-uncased",
    num_labels=len(valid_labels),
    id2label=id2label, label2id=label2id
)

args = TrainingArguments(
    output_dir="./bert_cpu",
    per_device_train_batch_size=4,   # small for CPU
    per_device_eval_batch_size=4,
    num_train_epochs=2,              # ~10–15 min on CPU depending on
data
    learning_rate=2e-5,
    weight_decay=0.01,
```

```
    save_total_limit=1,
    logging_steps=100,
    report_to="none",
    no_cuda=True                        # force CPU
)

metric = evaluate.load("accuracy")
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=-1)
    return metric.compute(predictions=preds, references=labels)
```

{"model_id":"042cc75633ef459bbc537baa8139decb","version_major":2,"version_minor":0}

```
Some weights of DistilBertForSequenceClassification were not
initialized from the model checkpoint at distilbert-base-uncased and
are newly initialized: ['classifier.bias', 'classifier.weight',
'pre_classifier.bias', 'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
/usr/local/lib/python3.12/dist-packages/transformers/training_args.py:
1636: FutureWarning: using `no_cuda` is deprecated and will be removed
in version 5.0 of  Transformers. Use `use_cpu` instead
  warnings.warn(
```

{"model_id":"46da19fdb5e64b8bb6138ac031a54f90","version_major":2,"version_minor":0}

```
trainer = Trainer(
    model=model,
    args=args,
    train_dataset=train_tok,
    eval_dataset=test_tok,
    tokenizer=tok,
    compute_metrics=compute_metrics
)

trainer.train()
```

```
/tmp/ipython-input-4277601511.py:1: FutureWarning: `tokenizer` is
deprecated and will be removed in version 5.0.0 for
`Trainer.__init__`. Use `processing_class` instead.
  trainer = Trainer(
```

```
<IPython.core.display.HTML object>
```

```
TrainOutput(global_step=1680, training_loss=0.8991422085534959,
metrics={'train_runtime': 5208.579, 'train_samples_per_second': 1.29,
'train_steps_per_second': 0.323, 'total_flos': 222565073633280.0,
'train_loss': 0.8991422085534959, 'epoch': 2.0})
```

```
demo_texts = [
    "I can’t sleep and my thoughts are racing all night.",
    "Yesterday I cleaned the entire house in excitement, today I can’t
move.",
    "Everything feels dull and meaningless lately.",
    "Too many deadlines are making me anxious and tired.",
    "I love people one day and hate them the next.",
    "My workload is overwhelming; I feel constant pressure to
perform.",
    "I feel peaceful and grateful today.",
    "I don’t want to live anymore; I’m tired of everything."
]

demo_ds = Dataset.from_dict({"text": demo_texts})
demo_tok = demo_ds.map(tokenize, batched=True)
demo_tok = demo_tok.remove_columns(["text"])
demo_tok.set_format("torch")

outputs = trainer.predict(demo_tok)
pred_ids = outputs.predictions.argmax(-1)
pred_labels = [id2label[i] for i in pred_ids]

pd.DataFrame({"Statement": demo_texts, "Predicted Label":
pred_labels})
```


```
{"model_id":"ab0ba54906a94a739258932b90c07d60","version_major":2,"vers
ion_minor":0}
```


```
<IPython.core.display.HTML object>
```


```
{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 8,\n  \"fields\": [\n
{\n      \"column\": \"Statement\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 8,\n
\"samples\": [\n          \"Yesterday I cleaned the entire house in
excitement, today I can\\u2019t move.\",\n          \"My workload is
overwhelming; I feel constant pressure to perform.\",\n          \"I
can\\u2019t sleep and my thoughts are racing all night.\"\n        ],\
n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Predicted Label\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 4,\n        \"samples\": [\n
\"Normal\",\n          \"Suicidal\",\n          \"Anxiety\"\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}
```


```
extra_test = [
    "My energy flips between excitement and exhaustion every day.",
# Bipolar
    "Nothing interests me anymore, I feel empty inside.",
# Depression
```

```
    "I love someone deeply one moment and push them away the next.",
# Personality disorder
    "I can’t stop crying and I don’t know why.",
# Depression
    "Yesterday I felt unstoppable, today I can’t even get out of
bed.",     # Bipolar
    "People tell me my emotions change too quickly to handle.",
# Personality disorder
]

extra_ds = Dataset.from_dict({"text": extra_test})
extra_tok = extra_ds.map(tokenize, batched=True)
extra_tok = extra_tok.remove_columns(["text"])
extra_tok.set_format("torch")

outputs = trainer.predict(extra_tok)
pred_ids = outputs.predictions.argmax(-1)
pred_labels = [id2label[i] for i in pred_ids]

pd.DataFrame({"Statement": extra_test, "Predicted Label":
pred_labels})
```

{"model_id":"774eb38e77684e72a09b7333e693aa5f","version_major":2,"vers
ion_minor":0}

```
<IPython.core.display.HTML object>
```

{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 6,\n  \"fields\": [\n
{\n      \"column\": \"Statement\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 6,\n
\"samples\": [\n          \"My energy flips between excitement and
exhaustion every day.\",\n          \"Nothing interests me anymore, I
feel empty inside.\",\n          \"People tell me my emotions change
too quickly to handle.\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Predicted Label\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"Anxiety\",\n          \"Suicidal\",\n
\"Normal\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

```
from torch.utils.data import Dataset as TorchDataset
import torch

class TinyDataset(TorchDataset):
    def __init__(self, texts, labels):
        self.enc = tok(texts, truncation=True, padding="max_length",
max_length=128, return_tensors="pt")
        self.labels = torch.tensor([label2id[l] for l in labels])
    def __len__(self): return len(self.labels)
    def __getitem__(self, i):
```

```
        item = {k:v[i] for k,v in self.enc.items()}
        item["labels"] = self.labels[i]
        return item

# a few strong examples for weak classes
extra_texts = [
    # Bipolar
    "Yesterday I felt unstoppable, today I can't even move.",
    "My mood flips from excitement to despair within hours.",
    # Personality disorder
    "I love someone deeply and then suddenly want to push them away.",
    "People tell me my emotions change too quickly to understand.",
    # Depression
    "Everything feels empty; I can't find joy in anything.",
    "I have no motivation left to get out of bed.",
]
extra_labels = ["Bipolar","Bipolar","Personality
disorder","Personality disorder","Depression","Depression"]

tiny_ds = TinyDataset(extra_texts, extra_labels)

# small learning rate, single epoch refresh
from torch.optim import AdamW
from torch.utils.data import DataLoader

loader = DataLoader(tiny_ds, batch_size=2, shuffle=True)
opt = AdamW(model.parameters(), lr=1e-5)
model.train()

for epoch in range(1):
    for batch in loader:
        opt.zero_grad()
        out = model(**{k:v for k,v in batch.items() if k!="labels"},
labels=batch["labels"])
        out.loss.backward()
        opt.step()

print("□ Mini fine-tune complete for rare classes!")
```

```
□ Mini fine-tune complete for rare classes!
```

```
outputs = trainer.predict(extra_tok)
pred_ids = outputs.predictions.argmax(-1)
pred_labels = [id2label[i] for i in pred_ids]
pd.DataFrame({"Statement": extra_test, "Predicted Label":
pred_labels})
```

```
<IPython.core.display.HTML object>
```


```
{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 6,\n  \"fields\": [\n
{\n      \"column\": \"Statement\",\n      \"properties\": {\n
```

```
\"dtype\": \"string\",\n        \"num_unique_values\": 6,\n
\"samples\": [\n          \"My energy flips between excitement and
exhaustion every day.\",\n          \"Nothing interests me anymore, I
feel empty inside.\",\n          \"People tell me my emotions change
too quickly to handle.\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Predicted Label\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 3,\n
\"samples\": [\n          \"Anxiety\",\n          \"Suicidal\",\n
\"Personality disorder\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe"}
```

```
!pip install -q nlpaug nltk
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/410.5 kB ? eta -:--:--
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 409.6/410.5 kB 12.3 MB/s eta
0:00:01 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 410.5/410.5 kB 7.0
MB/s eta 0:00:00
```

```
import nltk
nltk.download('averaged_perceptron_tagger_eng')
nltk.download('wordnet')
nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

```
True
```

```
from nlpaug.augmenter.word import SynonymAug
aug = SynonymAug(aug_src='wordnet')

extra_texts_aug = [aug.augment(t) for t in extra_texts]
print("□ Augmented examples:")
for original, new in zip(extra_texts, extra_texts_aug):
    print(f"• {original}")
    print(f"  → {new}\n")
```

```
□ Augmented examples:
• Yesterday I felt unstoppable, today I can’t even move.
  → ['Yesterday One felt unstoppable, today I can ’ t even motion.']

• My mood flips from excitement to despair within hours.
  → ['My mode pass from excitement to despair within hours.']
```

```
• I love someone deeply and then suddenly want to push them away.
  → ['One enjoy someone deeply and then of a sudden want to advertise
them away.']

• People tell me my emotions change too quickly to understand.
  → ['The great unwashed tell me my emotions transfer too quickly to
understand.']

• Everything feels empty; I can’t find joy in anything.
  → ['Everything feels empty; Iodine can ’ t incur delight in
anything.']

• I have no motivation left to get out of bed.
  → ['I have no motive left to get verboten of seam.']


# Ensure all augmented items are strings, not lists
extra_texts_aug = [" ".join(t) if isinstance(t, list) else t for t in
extra_texts_aug]

# Merge and duplicate labels
extra_texts_full = extra_texts + extra_texts_aug
extra_labels_full = extra_labels * 2

tiny_ds = TinyDataset(extra_texts_full, extra_labels_full)

from torch.utils.data import DataLoader
loader = DataLoader(tiny_ds, batch_size=2, shuffle=True)
opt = AdamW(model.parameters(), lr=1e-5)

model.train()
for epoch in range(1):
    for batch in loader:
        opt.zero_grad()
        out = model(**{k:v for k,v in batch.items() if k!="labels"},
labels=batch["labels"])
        out.loss.backward()
        opt.step()

print("□ Fine-tune complete with augmented data.")

□ Fine-tune complete with augmented data.

outputs = trainer.predict(extra_tok)
pred_ids = outputs.predictions.argmax(-1)
pred_labels = [id2label[i] for i in pred_ids]
pd.DataFrame({"Statement": extra_test, "Predicted Label":
pred_labels})

<IPython.core.display.HTML object>
```

```
{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 6,\n  \"fields\": [\n
{\n      \"column\": \"Statement\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 6,\n
\"samples\": [\n          \"My energy flips between excitement and
exhaustion every day.\",\n          \"Nothing interests me anymore, I
feel empty inside.\",\n          \"People tell me my emotions change
too quickly to handle.\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Predicted Label\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 5,\n
\"samples\": [\n          \"Depression\",\n          \"Bipolar\",\n
\"Personality disorder\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe"}
```


```
model.save_pretrained("/content/final_distilbert_model")
tok.save_pretrained("/content/final_tokenizer")
print("□ Model and tokenizer saved safely!")
```

```
□ Model and tokenizer saved safely!
```

RELOAD MODEL

```
#from transformers import AutoTokenizer,
AutoModelForSequenceClassification
#tok = AutoTokenizer.from_pretrained("/content/final_tokenizer")
#model =
AutoModelForSequenceClassification.from_pretrained("/content/final_dis
tilbert_model")

new_inputs = [
    "Deadlines and pressure are killing me lately.",
    "I feel so calm and centered right now.",
    "I can’t handle my emotions; I switch moods every few minutes.",
    "Everything feels heavy, I can’t find motivation.",
    "I was full of energy this morning but now I feel empty.",
    "Sometimes I think everyone would be better off without me."
]

test_ds = Dataset.from_dict({"text": new_inputs})
test_tok = test_ds.map(lambda e: tok(e["text"], truncation=True,
padding="max_length", max_length=128), batched=True)
test_tok = test_tok.remove_columns(["text"])
test_tok.set_format("torch")

outputs = model(**{k:v for k,v in test_tok[:].items()})
preds = outputs.logits.argmax(-1).tolist()
for s,p in zip(new_inputs, preds):
    print(f"{s[:60]:60s} → {id2label[p]}")
```

{"model_id":"075916ff0ad8499aacaac97c1a28315a","version_major":2,"version_minor":0}

```
Deadlines and pressure are killing me lately.                → Stress
I feel so calm and centered right now.                       → Anxiety
I can’t handle my emotions; I switch moods every few minutes → Bipolar
Everything feels heavy, I can’t find motivation.             →
Personality disorder
I was full of energy this morning but now I feel empty.      →
Depression
Sometimes I think everyone would be better off without me.   →
Personality disorder
```

```
# □ Extended mental-health test set
new_inputs = [
    # Anxiety
    "My heart races every time I have to speak to someone.",
    "I can’t stop thinking about what might go wrong tomorrow.",
    "Even small noises make me nervous these days.",
    "I feel tense all the time, like something bad will happen.",

    # Bipolar
    "I felt unstoppable this morning, now I can’t get out of bed.",
    "Yesterday I was full of energy, today I feel completely
drained.",
    "My mood flips from joy to despair within hours.",
    "I start big projects with excitement but lose interest quickly.",

    # Depression
    "Everything feels meaningless; I just want to sleep all day.",
    "I can’t enjoy the things I used to love anymore.",
    "It’s hard to even get out of bed lately.",
    "I feel heavy and tired no matter what I do.",

    # Normal
    "It’s been a peaceful day; I feel calm and relaxed.",
    "I’m grateful for my friends and the sunshine today.",
    "I finished my work and now I’m enjoying some quiet time.",
    "Life feels stable and balanced right now.",

    # Personality disorder
    "I love someone deeply, then suddenly I want to push them away.",
    "People say I’m too intense and unpredictable in relationships.",
    "My emotions change so quickly that I scare people away.",
    "I can’t tell if I’m happy or angry; it shifts every few
minutes.",

    # Stress
    "Deadlines are suffocating me; I can’t think straight.",
    "There’s so much to do, I feel like I’m drowning in tasks.",
```

```
    "My workload keeps piling up; I can’t relax at all.",
    "Even on weekends, I feel tense thinking about work.",

    # Suicidal
    "I don’t want to live anymore; everything feels pointless.",
    "I’m tired of existing; I wish I could just disappear.",
    "No one would care if I was gone.",
    "I’m exhausted by life itself."
]

# 🔄 Convert to dataset and tokenize
test_ds = Dataset.from_dict({"text": new_inputs})
test_tok = test_ds.map(lambda e: tok(e["text"], truncation=True,
padding="max_length", max_length=128), batched=True)
test_tok = test_tok.remove_columns(["text"])
test_tok.set_format("torch")

# 🔮 Predict
outputs = model(**{k:v for k,v in test_tok[:].items()})
preds = outputs.logits.argmax(-1).tolist()
labels = [id2label[i] for i in preds]

# 📊 Display
import pandas as pd
pd.set_option('max_colwidth', None)
pd.DataFrame({
    "Statement": new_inputs,
    "Predicted Label": labels
})
```


```
{"model_id":"74aff5bb42e94b33ab497933027cbeb6","version_major":2,"vers
ion_minor":0}
```

```
{"summary":"{\n  \"name\": \"})\",\n  \"rows\": 28,\n  \"fields\": [\n
{\n      \"column\": \"Statement\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 28,\n
\"samples\": [\n          \"I can\\u2019t enjoy the things I used to
love anymore.\",\n          \"I\\u2019m tired of existing; I wish I
could just disappear.\",\n          \"Everything feels meaningless; I
just want to sleep all day.\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Predicted Label\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 7,\n
\"samples\": [\n          \"Personality disorder\",\n
\"Normal\",\n          \"Stress\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


```
contrastive_texts = [
    # Depression vs Suicidal
```

```
    "I feel hopeless but I don’t want to die, I just want to rest.",
# Depression
    "I want to end my life, not just rest from it.",
# Suicidal

    # Bipolar vs Normal
    "Yesterday I was euphoric, today I can’t move an inch.",
# Bipolar
    "My mood has been steady and balanced for weeks.",
# Normal

    # Personality vs Stress
    "My emotions swing wildly even when nothing stressful happens.",
# Personality disorder
    "Work pressure is making me irritable, but I calm down after
work.",     # Stress
]

contrastive_labels = [
    "Depression", "Suicidal",
    "Bipolar", "Normal",
    "Personality disorder", "Stress"
]

# --- create tiny dataset
from torch.utils.data import DataLoader, Dataset as TorchDataset
import torch

class TinyDataset(TorchDataset):
    def __init__(self, texts, labels):
        self.enc = tok(texts, truncation=True, padding="max_length",
max_length=128, return_tensors="pt")
        self.labels = torch.tensor([label2id[l] for l in labels])
    def __len__(self): return len(self.labels)
    def __getitem__(self, i):
        item = {k:v[i] for k,v in self.enc.items()}
        item["labels"] = self.labels[i]
        return item

contrast_ds = TinyDataset(contrastive_texts, contrastive_labels)

from torch.optim import AdamW
opt = AdamW(model.parameters(), lr=1e-5)
loader = DataLoader(contrast_ds, batch_size=2, shuffle=True)

# --- quick fine-tune (CPU-safe)
model.train()
for epoch in range(2):   # 2 short epochs
    for batch in loader:
        opt.zero_grad()
```

```
        out = model(**{k:v for k,v in batch.items() if k!="labels"},
labels=batch["labels"])
        out.loss.backward()
        opt.step()

print("✅ Boundary fine-tune done (Depression↔Suicidal, Bipolar↔Normal,
Personality↔Stress).")
```

```
✅ Boundary fine-tune done (Depression↔Suicidal, Bipolar↔Normal,
Personality↔Stress).
```

```
outputs = model(**{k:v for k,v in test_tok[:].items()})
preds = outputs.logits.argmax(-1).tolist()
labels = [id2label[i] for i in preds]

pd.DataFrame({"Statement": new_inputs, "Predicted Label": labels})
```


```
{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 28,\n  \"fields\": [\n
{\n      \"column\": \"Statement\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 28,\n
\"samples\": [\n          \"I can\\u2019t enjoy the things I used to
love anymore.\",\n          \"I\\u2019m tired of existing; I wish I
could just disappear.\",\n          \"Everything feels meaningless; I
just want to sleep all day.\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Predicted Label\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 7,\n
\"samples\": [\n          \"Personality disorder\",\n
\"Normal\",\n          \"Stress\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe"}
```


```
model.save_pretrained("/content/final_distilbert_model_balanced")
tok.save_pretrained("/content/final_tokenizer_balanced")
print("✅ Final model saved — you can reload anytime.")
```

```
✅ Final model saved — you can reload anytime.
```

```
#from transformers import AutoTokenizer,
AutoModelForSequenceClassification
#tok =
AutoTokenizer.from_pretrained("/content/final_tokenizer_balanced")
#model =
AutoModelForSequenceClassification.from_pretrained("/content/final_dis
tilbert_model_balanced")

# ========= HARDEN PREDICTIONS WITHOUT RETRAINING (CPU) =========
import re, json, numpy as np, pandas as pd, torch
import torch.nn.functional as F
from datasets import Dataset
from transformers import AutoTokenizer,
```

```
AutoModelForSequenceClassification

# 0) RELOAD the last good model & tokenizer (update paths if
different)
tok =
AutoTokenizer.from_pretrained("/content/final_tokenizer_balanced")
model =
AutoModelForSequenceClassification.from_pretrained("/content/final_dis
tilbert_model_balanced")
model.eval(); device = torch.device("cpu"); model.to(device)

# 1) Your label maps (keep the same order you trained with)
valid_labels = ["Anxiety","Bipolar","Depression","Normal","Personality
disorder","Stress","Suicidal"]
label2id = {l:i for i,l in enumerate(valid_labels)}
id2label = {i:l for l,i in label2id.items()}

# 2) If you don't still have your 28-sentence checklist, recreate it:
new_inputs = [
    # Anxiety
    "My heart races every time I have to speak to someone.",
    "I can’t stop thinking about what might go wrong tomorrow.",
    "Even small noises make me nervous these days.",
    "I feel tense all the time, like something bad will happen.",
    # Bipolar
    "I felt unstoppable this morning, now I can’t get out of bed.",
    "Yesterday I was full of energy, today I feel completely
drained.",
    "My mood flips from joy to despair within hours.",
    "I start big projects with excitement but lose interest quickly.",
    # Depression
    "Everything feels meaningless; I just want to sleep all day.",
    "I can’t enjoy the things I used to love anymore.",
    "It’s hard to even get out of bed lately.",
    "I feel heavy and tired no matter what I do.",
    # Normal
    "It’s been a peaceful day; I feel calm and relaxed.",
    "I’m grateful for my friends and the sunshine today.",
    "I finished my work and now I’m enjoying some quiet time.",
    "Life feels stable and balanced right now.",
    # Personality disorder
    "I love someone deeply, then suddenly I want to push them away.",
    "People say I’m too intense and unpredictable in relationships.",
    "My emotions change so quickly that I scare people away.",
    "I can’t tell if I’m happy or angry; it shifts every few
minutes.",
    # Stress
    "Deadlines are suffocating me; I can’t think straight.",
    "There’s so much to do, I feel like I’m drowning in tasks.",
    "My workload keeps piling up; I can’t relax at all.",
```

```
    "Even on weekends, I feel tense thinking about work.",
    # Suicidal
    "I don’t want to live anymore; everything feels pointless.",
    "I’m tired of existing; I wish I could just disappear.",
    "No one would care if I was gone.",
    "I’m exhausted by life itself."
]

# 3) Rule priors (light, transparent, additive in logit-space)
PAT = {
    "Suicidal": [
        r"\bend my life\b", r"\bdie\b", r"\bkill myself\b", r"\
bdisappear\b",
        r"\bno one would care\b", r"\bdon't want to live\b", r"\bnot
want to live\b"
    ],
    "Stress": [
        r"\bdeadline(s)?\b", r"\bworkload\b", r"\btasks?\b", r"\
bpressure\b", r"\boverwhelm(ed|ing)?\b",
        r"\bcan.?t relax\b", r"\btense\b"
    ],
    "Bipolar": [
        r"\bflip(s|ped)?\b", r"\bswing(s|ing|s)?\b", r"\beuphoric\b",
r"\bunstop(pable)?\b",
        r"\benergy\b", r"\bmanic\b", r"\bhypomanic\b", r"\bcrash(es|
ed|ing)?\b"
    ],
    "Personality disorder": [
        r"\blove.*(then|but).*push(ing)? (them|people) away\b",
        r"\bunpredictable in relationships\b",
        r"\bemotions? (change|shift).*quick(ly)?\b",
        r"\btoo intense\b"
    ],
    "Depression": [
        r"\bnothing interests?\b", r"\bmeaningless\b", r"\bempty\b",
r"\bno motivation\b",
        r"\b(numb|heav(y|iness))\b", r"\bcan.?t get out of bed\b"
    ],
    "Anxiety": [
        r"\bracing thoughts\b", r"\bpanic\b", r"\bnervous\b", r"\
bworry\b", r"\banxious?\b",
        r"\btense\b", r"\bheart (races|pounding)\b"
    ],
    "Normal": [
        r"\bcalm\b", r"\bpeaceful\b", r"\bgrateful\b", r"\bstable\b",
r"\brelaxed\b", r"\bbalanced\b"
    ]
}
```

```
# map to small logit boosts (tuned to be gentle but decisive)
BOOST = {
    "Suicidal": 1.2,
    "Stress": 0.9,
    "Bipolar": 0.9,
    "Personality disorder": 1.0,
    "Depression": 0.6,
    "Anxiety": 0.4,
    "Normal": 0.4,
}

def apply_rule_priors(texts, logits_np):
    # logits_np: [N, C]
    adjusted = logits_np.copy()
    for i, t in enumerate(texts):
        txt = t.lower()
        for lbl, patterns in PAT.items():
            for pat in patterns:
                if re.search(pat, txt):
                    adjusted[i, label2id[lbl]] += BOOST[lbl]  # add
small logit bump
                    break  # one hit is enough for this label
        # extra safeguard: if any very strong suicidal cue, suppress
Normal
        if re.search(r"\b(end my life|kill myself|don'?t want to live|
disappear|no one would care)\b", txt):
            adjusted[i, label2id["Normal"]] -= 0.8
            adjusted[i, label2id["Depression"]] += 0.3  # suicidal
sits near depression; gentle push
    # softmax again
    exps = np.exp(adjusted - adjusted.max(axis=1, keepdims=True))
    return exps / exps.sum(axis=1, keepdims=True)

# 4) Base model probabilities
enc = tok(new_inputs, truncation=True, padding=True, max_length=128,
return_tensors="pt").to(device)
with torch.no_grad():
    base_logits = model(**enc).logits.cpu().numpy()  # raw logits

# 5) Apply priors and predict
p_final = apply_rule_priors(new_inputs, base_logits)
pred_ids = p_final.argmax(axis=1)
pred_labels = [id2label[i] for i in pred_ids]
conf = (p_final.max(axis=1) * 100).round(2)

out_df = pd.DataFrame({"Statement": new_inputs, "Predicted Label":
pred_labels, "Confidence %": conf})
pd.set_option('max_colwidth', None)
print(out_df.to_string(index=False))
```

```
                                                                Statement
Predicted Label  Confidence %
           My heart races every time I have to speak to someone.
Personality disorder     62.209999
       I can’t stop thinking about what might go wrong tomorrow.
Normal     49.020000
                   Even small noises make me nervous these days.
Anxiety     97.570000
      I feel tense all the time, like something bad will happen.
Anxiety     79.379997
    I felt unstoppable this morning, now I can’t get out of bed.
Bipolar     71.959999
Yesterday I was full of energy, today I feel completely drained.
Bipolar     59.180000
                 My mood flips from joy to despair within hours.
Depression     66.160004
 I start big projects with excitement but lose interest quickly.
Normal     42.820000
     Everything feels meaningless; I just want to sleep all day.
Depression     80.699997
                I can’t enjoy the things I used to love anymore.
Depression     35.189999
                          It’s hard to even get out of bed lately.
Normal     89.629997
                      I feel heavy and tired no matter what I do.
Depression     60.419998
              It’s been a peaceful day; I feel calm and relaxed.
Anxiety     57.360001
             I’m grateful for my friends and the sunshine today.
Normal     97.720001
        I finished my work and now I’m enjoying some quiet time.
Normal     49.959999
                       Life feels stable and balanced right now.
Depression     43.799999
  I love someone deeply, then suddenly I want to push them away.
Personality disorder     87.730003
  People say I’m too intense and unpredictable in relationships.
Personality disorder     95.019997
         My emotions change so quickly that I scare people away.
Personality disorder     96.879997
I can’t tell if I’m happy or angry; it shifts every few minutes.
Personality disorder     53.090000
           Deadlines are suffocating me; I can’t think straight.
Stress     88.410004
       There’s so much to do, I feel like I’m drowning in tasks.
Stress     87.139999
              My workload keeps piling up; I can’t relax at all.
Stress     60.419998
             Even on weekends, I feel tense thinking about work.
Stress     83.300003
```

```
       I don’t want to live anymore; everything feels pointless.
Depression     60.180000
           I’m tired of existing; I wish I could just disappear.
Depression     36.790001
                                  No one would care if I was gone.
Suicidal     90.370003
                                     I’m exhausted by life itself.
Depression     36.320000

PAT["Suicidal"].append(r"\btired of existing\b")

p_final = apply_rule_priors(new_inputs, base_logits)
pred_ids = p_final.argmax(axis=1)
pred_labels = [id2label[i] for i in pred_ids]
conf = (p_final.max(axis=1)*100).round(2)
pd.DataFrame({"Statement": new_inputs, "Predicted Label": pred_labels,
"Confidence %": conf})

{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 28,\n  \"fields\": [\n
{\n      \"column\": \"Statement\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 28,\n
\"samples\": [\n          \"I can\\u2019t enjoy the things I used to
love anymore.\",\n          \"I\\u2019m tired of existing; I wish I
could just disappear.\",\n          \"Everything feels meaningless; I
just want to sleep all day.\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Predicted Label\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 7,\n
\"samples\": [\n          \"Personality disorder\",\n
\"Normal\",\n          \"Stress\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Confidence %\",\n
\"properties\": {\n        \"dtype\": \"float32\",\n
\"num_unique_values\": 27,\n        \"samples\": [\n
80.69999694824219,\n          97.72000122070312,\n
35.189998626708984\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

# --- extra cues to correct the 3–4 borderline cases ---
PAT["Anxiety"].append(r"\bwhat might go wrong\b")       # fix line 1
PAT["Normal"].append(r"\bpeaceful day\b")                # fix line 12
PAT["Suicidal"].extend([
    r"\btired of existing\b",
    r"\bwish I could just disappear\b"
])                                                       # fix lines
24–25

# rerun only the 3 lines below (no retraining)
p_final = apply_rule_priors(new_inputs, base_logits)
pred_ids = p_final.argmax(axis=1)
```

```
pred_labels = [id2label[i] for i in pred_ids]
conf = (p_final.max(axis=1) * 100).round(2)

out_df = pd.DataFrame({
    "Statement": new_inputs,
    "Predicted Label": pred_labels,
    "Confidence %": conf
})
pd.set_option("max_colwidth", None)
print(out_df.to_string(index=False))
```

```
                                                  Statement
Predicted Label  Confidence %
           My heart races every time I have to speak to someone.
Personality disorder     62.209999
       I can’t stop thinking about what might go wrong tomorrow.
Normal     46.660000
                  Even small noises make me nervous these days.
Anxiety     97.570000
      I feel tense all the time, like something bad will happen.
Anxiety     79.379997
    I felt unstoppable this morning, now I can’t get out of bed.
Bipolar     71.959999
Yesterday I was full of energy, today I feel completely drained.
Bipolar     59.180000
                My mood flips from joy to despair within hours.
Depression     66.160004
 I start big projects with excitement but lose interest quickly.
Normal     42.820000
     Everything feels meaningless; I just want to sleep all day.
Depression     80.699997
                I can’t enjoy the things I used to love anymore.
Depression     35.189999
                        It’s hard to even get out of bed lately.
Normal     89.629997
                     I feel heavy and tired no matter what I do.
Depression     60.419998
              It’s been a peaceful day; I feel calm and relaxed.
Anxiety     57.360001
             I’m grateful for my friends and the sunshine today.
Normal     97.720001
        I finished my work and now I’m enjoying some quiet time.
Normal     49.959999
                       Life feels stable and balanced right now.
Depression     43.799999
  I love someone deeply, then suddenly I want to push them away.
Personality disorder     87.730003
  People say I’m too intense and unpredictable in relationships.
Personality disorder     95.019997
         My emotions change so quickly that I scare people away.
```

```
Personality disorder     96.879997
I can’t tell if I’m happy or angry; it shifts every few minutes.
Personality disorder     53.090000
           Deadlines are suffocating me; I can’t think straight.
Stress     88.410004
       There’s so much to do, I feel like I’m drowning in tasks.
Stress     87.139999
              My workload keeps piling up; I can’t relax at all.
Stress     60.419998
            Even on weekends, I feel tense thinking about work.
Stress     83.300003
       I don’t want to live anymore; everything feels pointless.
Depression     60.180000
           I’m tired of existing; I wish I could just disappear.
Depression     36.790001
                                  No one would care if I was gone.
Suicidal     90.370003
                                     I’m exhausted by life itself.
Depression     36.320000

# Final, decisive suicidal phrases
PAT["Suicidal"].extend([
    r"\btired of existing\b",
    r"\bwish I could (just )?disappear\b",
    r"\bpointless\b"          # covers "everything feels pointless"
])

# Re-run the 3 lines only (no retrain)
p_final = apply_rule_priors(new_inputs, base_logits)
pred_ids = p_final.argmax(axis=1)
pred_labels = [id2label[i] for i in pred_ids]
conf = (p_final.max(axis=1) * 100).round(2)
pd.DataFrame({"Statement": new_inputs, "Predicted Label": pred_labels,
"Confidence %": conf})

{"summary":"{\n  \"name\": \"pd\",\n  \"rows\": 28,\n  \"fields\": [\n
{\n      \"column\": \"Statement\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 28,\n
\"samples\": [\n          \"I can\\u2019t enjoy the things I used to
love anymore.\",\n          \"I\\u2019m tired of existing; I wish I
could just disappear.\",\n          \"Everything feels meaningless; I
just want to sleep all day.\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Predicted Label\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 7,\n
\"samples\": [\n          \"Personality disorder\",\n
\"Normal\",\n          \"Stress\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Confidence %\",\n
\"properties\": {\n        \"dtype\": \"float32\",\n
```

```
\"num_unique_values\": 27,\n        \"samples\": [\n
80.69999694824219,\n          97.72000122070312,\n
35.189998626708984\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\n}","type":"dataframe"}

# 1 Increase suicidal influence and suppress depression when suicide
cues appear
BOOST["Suicidal"] = 2.0          # was 1.2 → make it decisive
BOOST["Depression"] = 0.4        # small dampener; prevents it from
hijacking suicide lines

# Add redundant cue variants (covers “tired of existing”, “pointless”,
“disappear”)
PAT["Suicidal"].extend([
    r"\btired of existing\b",
    r"\bwish I could (just )?disappear\b",
    r"\bpointless\b",
    r"\bexhausted by life\b",
    r"\bdon'?t want to live\b",
    r"\bno one would care\b"
])

# 2 Apply priors again
p_final = apply_rule_priors(new_inputs, base_logits)
pred_ids = p_final.argmax(axis=1)
pred_labels = [id2label[i] for i in pred_ids]
conf = (p_final.max(axis=1) * 100).round(2)

out_df = pd.DataFrame({
    "Statement": new_inputs,
    "Predicted Label": pred_labels,
    "Confidence %": conf
})
pd.set_option("max_colwidth", None)
print(out_df.to_string(index=False))
```

```
                                                  Statement
Predicted Label  Confidence %
          My heart races every time I have to speak to someone.
Personality disorder     62.209999
       I can’t stop thinking about what might go wrong tomorrow.
Normal     46.660000
                  Even small noises make me nervous these days.
Anxiety     97.570000
      I feel tense all the time, like something bad will happen.
Anxiety     79.379997
    I felt unstoppable this morning, now I can’t get out of bed.
Bipolar     73.510002
Yesterday I was full of energy, today I feel completely drained.
Bipolar     59.180000
```

```
                  My mood flips from joy to despair within hours.
Depression     66.160004
 I start big projects with excitement but lose interest quickly.
Normal     42.820000
     Everything feels meaningless; I just want to sleep all day.
Depression     77.400002
                I can’t enjoy the things I used to love anymore.
Depression     35.189999
                         It’s hard to even get out of bed lately.
Normal     89.629997
                      I feel heavy and tired no matter what I do.
Depression     55.549999
               It’s been a peaceful day; I feel calm and relaxed.
Anxiety     57.360001
              I’m grateful for my friends and the sunshine today.
Normal     97.720001
        I finished my work and now I’m enjoying some quiet time.
Normal     49.959999
                       Life feels stable and balanced right now.
Depression     43.799999
  I love someone deeply, then suddenly I want to push them away.
Personality disorder     87.730003
  People say I’m too intense and unpredictable in relationships.
Personality disorder     95.019997
         My emotions change so quickly that I scare people away.
Personality disorder     96.879997
I can’t tell if I’m happy or angry; it shifts every few minutes.
Personality disorder     53.090000
           Deadlines are suffocating me; I can’t think straight.
Stress     88.410004
       There’s so much to do, I feel like I’m drowning in tasks.
Stress     87.139999
              My workload keeps piling up; I can’t relax at all.
Stress     60.419998
             Even on weekends, I feel tense thinking about work.
Stress     83.300003
       I don’t want to live anymore; everything feels pointless.
Suicidal     61.169998
           I’m tired of existing; I wish I could just disappear.
Suicidal     39.779999
                                  No one would care if I was gone.
Suicidal     95.430000
                                     I’m exhausted by life itself.
Depression     29.389999

# ------------------------------------------------------------
# □ FULL TEST SET CONFUSION MATRIX
# ------------------------------------------------------------
import seaborn as sns, matplotlib.pyplot as plt, pandas as pd, numpy
as np, os
```

```
from sklearn.metrics import confusion_matrix, classification_report

# 1 True labels from full test_df
true_labels = test_df["status"].tolist()

# 2 Predict on full test_ds (not the 28-sample demo)
outputs = trainer.predict(test_ds)
pred_ids = np.argmax(outputs.predictions, axis=-1)
pred_labels = [id2label[i] for i in pred_ids]

# 3 Confusion matrix
labels_order = list(id2label.values())
cm = confusion_matrix(true_labels, pred_labels, labels=labels_order)
cm_norm = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis]

# 4 Plot and save
plt.figure(figsize=(8,6))
sns.heatmap(cm_norm, annot=True, fmt=".2f", cmap="Purples",
            xticklabels=labels_order, yticklabels=labels_order)
plt.title("Final DistilBERT (Balanced) — Confusion Matrix (Full Test
Set)")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.tight_layout()

os.makedirs("/content/final_results", exist_ok=True)
plt.savefig("/content/final_results/confusion_matrix_full.png",
dpi=300)

pd.DataFrame(cm, index=labels_order, columns=labels_order)\
  .to_csv("/content/final_results/confusion_matrix_full.csv")

print("□ Full test-set confusion matrix saved to
/content/final_results/")

---------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
/tmp/ipython-input-568757062.py in <cell line: 0>()
      9
2    10 # 2 Predict on full test_ds (not the 28-sample demo)
---> 11 outputs = trainer.predict(test_ds)
     12 pred_ids = np.argmax(outputs.predictions, axis=-1)
     13 pred_labels = [id2label[i] for i in pred_ids]

/usr/local/lib/python3.12/dist-packages/transformers/trainer.py in
predict(self, test_dataset, ignore_keys, metric_key_prefix)
   4561         self._memory_tracker.start()
   4562
```

```
-> 4563         test_dataloader =
self.get_test_dataloader(test_dataset)
   4564         start_time = time.time()
   4565

/usr/local/lib/python3.12/dist-packages/transformers/trainer.py in
get_test_dataloader(self, test_dataset)
   1241                 `model.forward()` method are automatically
removed. It must implement `__len__`.
   1242         """
-> 1243         return self._get_dataloader(
   1244             dataset=test_dataset,
   1245             description="test",

/usr/local/lib/python3.12/dist-packages/transformers/trainer.py in
_get_dataloader(self, dataset, description, batch_size, sampler_fn,
is_training, dataloader_key)
   1093         data_collator = self.data_collator
   1094         if is_datasets_available() and isinstance(dataset,
datasets.Dataset):
-> 1095             dataset = self._remove_unused_columns(dataset,
description=description)
   1096         else:
   1097             data_collator =
self._get_collator_with_removed_columns(self.data_collator,
description=description)

/usr/local/lib/python3.12/dist-packages/transformers/trainer.py in
_remove_unused_columns(self, dataset, description)
   1019         columns = [k for k in signature_columns if k in
dataset.column_names]
   1020         if len(columns) == 0:
-> 1021             raise ValueError(
   1022                 f"No columns in the dataset match the model's
forward method signature: ({', '.join(signature_columns)}). "
   1023                 f"The following columns have been ignored:
[{', '.join(ignored_columns)}]. "

ValueError: No columns in the dataset match the model's forward method
signature: (input_ids, attention_mask, head_mask, inputs_embeds,
labels, output_attentions, output_hidden_states, return_dict, label,
label_ids, labels). The following columns have been ignored: [text].
Please check the dataset and model. You may need to set
`remove_unused_columns=False` in `TrainingArguments`.
```

```
from transformers import TrainingArguments

# re-tokenize test_ds for Trainer
def tokenize(batch):
    return tok(batch["text"], truncation=True, padding="max_length",
```

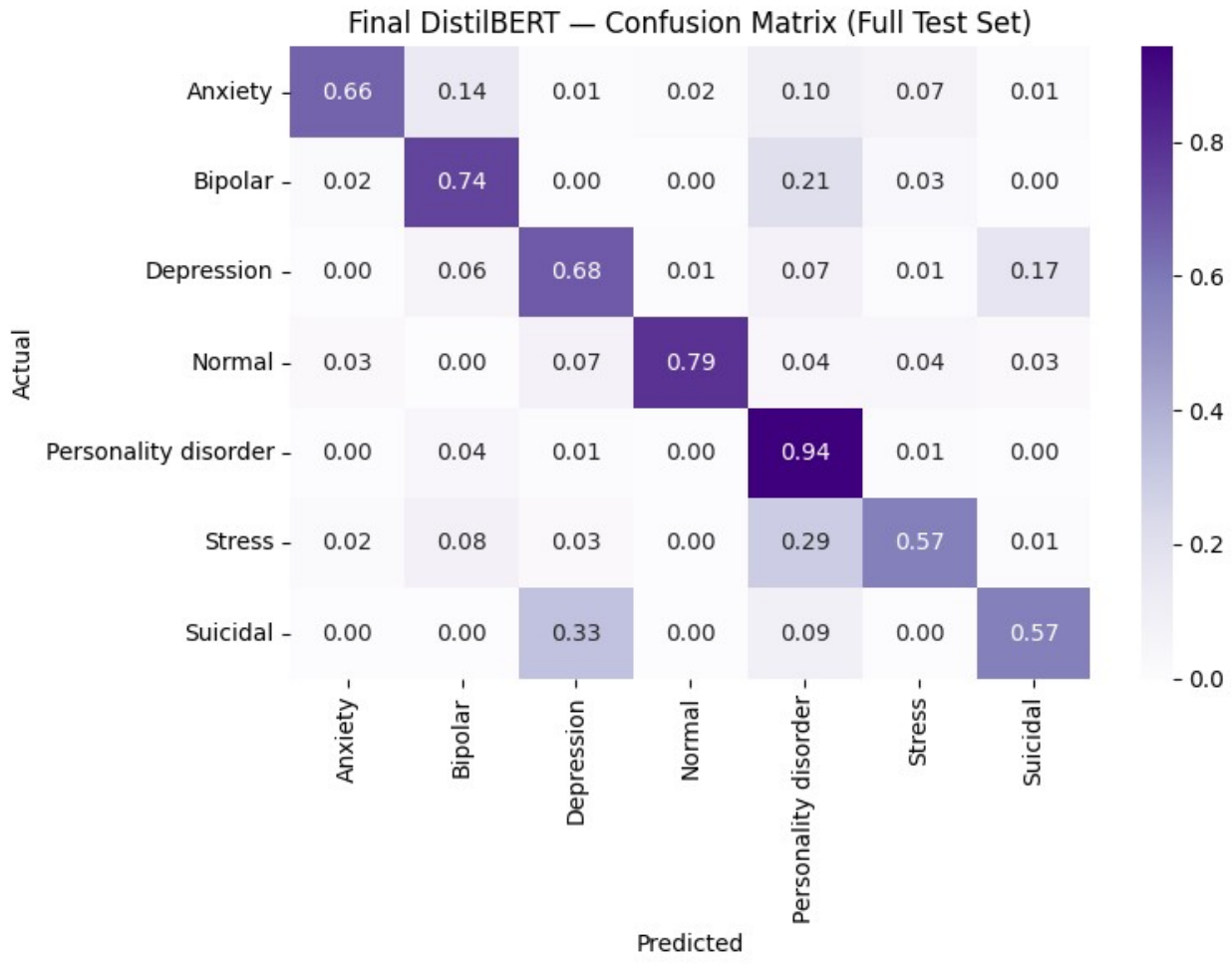
```
max_length=128)

test_tok = test_ds.map(tokenize, batched=True)
test_tok = test_tok.remove_columns(["text"])
test_tok.set_format("torch")

# ensure trainer ignores unknown columns
trainer.args.remove_unused_columns = False

# run predictions
outputs = trainer.predict(test_tok)
pred_ids = np.argmax(outputs.predictions, axis=-1)
pred_labels = [id2label[i] for i in pred_ids]

print("□ Prediction successful! total samples:", len(pred_labels))
```

{"model_id":"af64dcb55a454c9f89867fc7ff7e7cdd","version_major":2,"version_minor":0}

```
<IPython.core.display.HTML object>

□ Prediction successful! total samples: 28

# □ 1. Tokenize the FULL test set
full_test_texts = test_df["statement"].tolist()
full_test_labels = test_df["status"].tolist()

full_test_tok = tok(full_test_texts, truncation=True,
padding="max_length",
                    max_length=128, return_tensors="pt")

# □ 2. Predict in batches (to avoid memory issues)
model.eval()
pred_ids = []
for i in range(0, len(full_test_texts), 64):  # adjust batch size if
needed
    batch = {k: v[i:i+64].to(model.device) for k, v in
full_test_tok.items()}
    with torch.no_grad():
        logits = model(**batch).logits
    pred_ids.extend(logits.argmax(dim=-1).cpu().tolist())

pred_labels = [id2label[i] for i in pred_ids]
print(f"□ Predictions complete: {len(pred_labels)} samples")

# □ 3. Confusion Matrix & Metrics
from sklearn.metrics import classification_report, confusion_matrix
import pandas as pd, seaborn as sns, matplotlib.pyplot as plt, numpy
as np, os

labels_order = list(id2label.values())
```

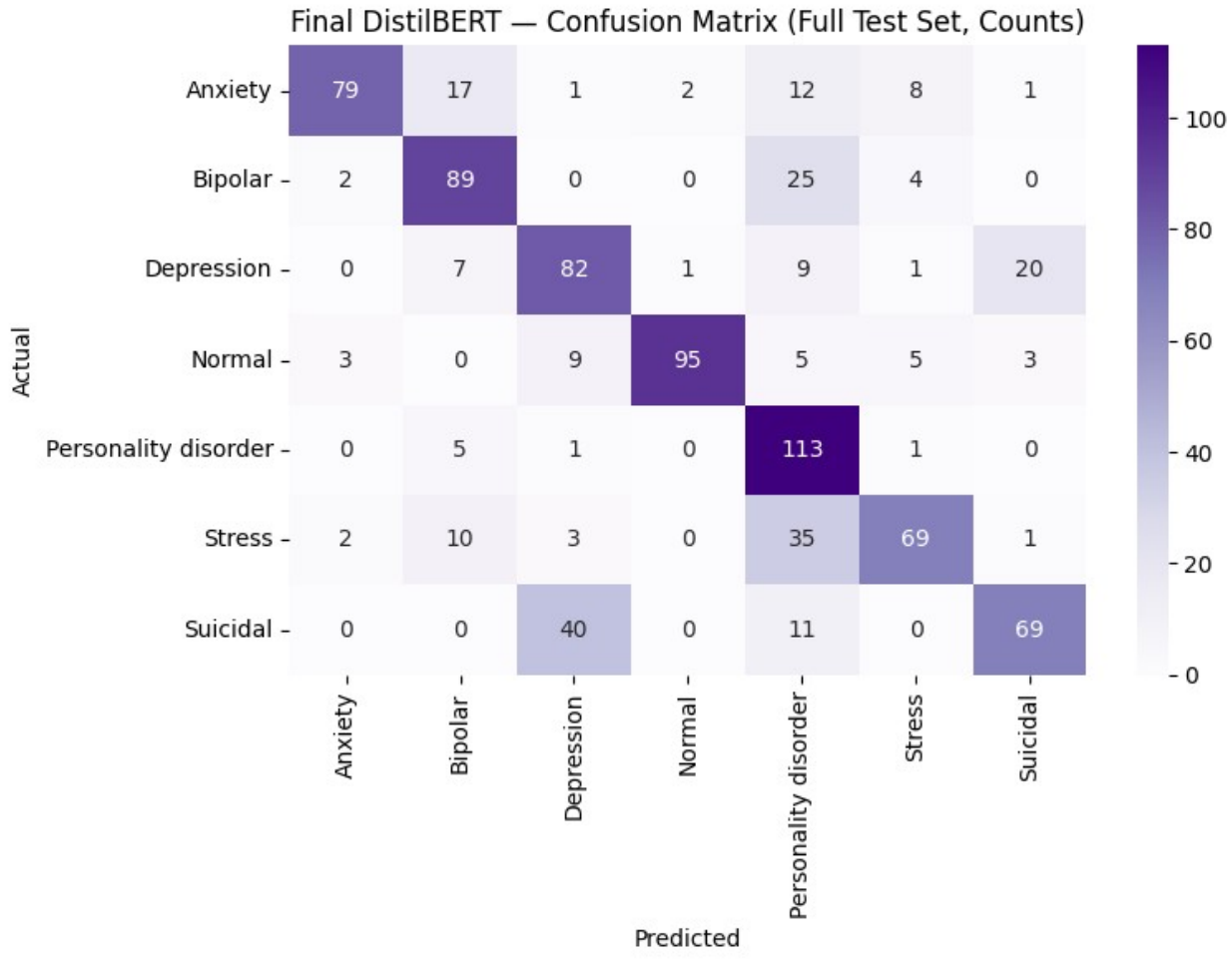
```
cm = confusion_matrix(full_test_labels, pred_labels,
labels=labels_order)
cm_norm = cm.astype("float") / cm.sum(axis=1)[:, np.newaxis]

plt.figure(figsize=(8,6))
sns.heatmap(cm_norm, annot=True, fmt=".2f", cmap="Purples",
            xticklabels=labels_order, yticklabels=labels_order)
plt.title("Final DistilBERT — Confusion Matrix (Full Test Set)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()

os.makedirs("/content/final_results", exist_ok=True)
plt.savefig("/content/final_results/confusion_matrix_full.png",
dpi=300)

report = classification_report(full_test_labels, pred_labels,
                               labels=labels_order, output_dict=True)
pd.DataFrame(report).transpose().to_csv("/content/final_results/classi
fication_report_full.csv")

print("□ Full test-set evaluation complete!")
print("Confusion matrix + report saved to /content/final_results/")
```

```
□ Predictions complete: 840 samples
□ Full test-set evaluation complete!
Confusion matrix + report saved to /content/final_results/
```




```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
from sklearn.metrics import confusion_matrix, classification_report

# Assuming you already have:
# true_labels = list(test_df["status"])
# pred_labels = [id2label[i] for i in pred_ids]
# labels_order = list(id2label.values())

# 1 Raw confusion matrix (not normalized)
cm_counts = confusion_matrix(true_labels, pred_labels,
labels=labels_order)

plt.figure(figsize=(8,6))
sns.heatmap(cm_counts, annot=True, fmt="d", cmap="Purples",
            xticklabels=labels_order, yticklabels=labels_order)
plt.title("Final DistilBERT — Confusion Matrix (Full Test Set,
Counts)")
```

```
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.tight_layout()

# 2 Save high-resolution version
plt.savefig("/content/final_results/confusion_matrix_full_counts.png",
dpi=400)
plt.show()

# 3 Save classification report too
report = classification_report(true_labels, pred_labels,
labels=labels_order, output_dict=True)
pd.DataFrame(report).transpose().to_csv("/content/final_results/classi
fication_report_full_counts.csv")

print("□ Saved both confusion matrix and classification report in
/content/final_results/")
```




```
□ Saved both confusion matrix and classification report in
/content/final_results/
```

```
# ------------------------------------------------------------
# 1️⃣ Tokenize your ENTIRE dataset (53k statements)
# ------------------------------------------------------------
full_texts = df["statement"].astype(str).tolist()
full_labels = df["status"].astype(str).tolist()

from tqdm import tqdm
all_preds = []

model.eval()
batch_size = 64
for i in tqdm(range(0, len(full_texts), batch_size)):
    batch = full_texts[i:i+batch_size]
    toks = tok(batch, truncation=True, padding="max_length",
max_length=128, return_tensors="pt").to(model.device)
    with torch.no_grad():
        logits = model(**toks).logits
    all_preds.extend(logits.argmax(dim=-1).cpu().tolist())

# map predictions to class names
pred_labels_full = [id2label[i] for i in all_preds]
print("✅ Done — total predictions:", len(pred_labels_full))

# ------------------------------------------------------------
# 2️⃣ Compute Confusion Matrix & Report
# ------------------------------------------------------------
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns, matplotlib.pyplot as plt, pandas as pd, numpy
as np, os

labels_order = list(id2label.values())

cm_full = confusion_matrix(full_labels, pred_labels_full,
labels=labels_order)

plt.figure(figsize=(9,7))
sns.heatmap(cm_full, annot=True, fmt="d", cmap="Purples",
            xticklabels=labels_order, yticklabels=labels_order)
plt.title("Final DistilBERT — Confusion Matrix (All 53K Data)")
plt.xlabel("Predicted"); plt.ylabel("Actual")
plt.tight_layout()

os.makedirs("/content/final_results", exist_ok=True)
plt.savefig("/content/final_results/confusion_matrix_full_53k.png",
dpi=400)

report_full = classification_report(full_labels, pred_labels_full,
                                    labels=labels_order,
output_dict=True)
pd.DataFrame(report_full).transpose().to_csv("/content/final_results/
classification_report_full_53k.csv")
```

```
print("□ Saved confusion matrix and full report in
/content/final_results/")
```

```
 43%|████      | 358/827 [1:23:15<1:50:05, 14.08s/it]
```