```
# ==========================================
# CELL 1: INSTALL & SETUP
# ==========================================
!pip install tensorflow numpy pandas matplotlib seaborn scikit-learn nltk nrclex
!python -m textblob.download_corpora
import nltk
nltk.download('stopwords')

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
import io
from nrclex import NRCLex
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, precision_score, recall_score, f1_score

import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Embedding, Bidirectional, LSTM, Dense, Dropout, Concatenate
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import BinaryCrossentropy

# --- RESEARCH PARAMETERS ---
K_CLIENTS = 10          # Federated Clients
SEQ_LENGTH = 100        # Fixed Sequence Length
VOCAB_SIZE = 10000      # Vocabulary Size
EMBED_DIM = 100         # Embedding Dimensions
HIDDEN_DIM = 64         # BiLSTM Hidden Units
GLOBAL_ROUNDS = 10      # FL Rounds
LOCAL_EPOCHS = 1        # Local Epochs per Round
BATCH_SIZE = 32

# Reproducibility
np.random.seed(42)
tf.random.set_seed(42)
print("Environment Ready.")
```

```
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.10.0)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.15.1)
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.5.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas) (2025.3)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.61.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
```

```
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.12/dist-packages (from rich->keras>=3.5.0->te
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.12/dist-packages (from rich->keras>=3.5.0->ter
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.12/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=
Building wheels for collected packages: nrclex
  Building wheel for nrclex (setup.py) ... done
  Created wheel for nrclex: filename=NRCLex-3.0.0-py3-none-any.whl size=43309 sha256=df7acda47aa91691d40becde5717db58cdc6368bf
  Stored in directory: /root/.cache/pip/wheels/1f/e8/d0/e3c3da0ef3b37ef4381dbf5c9401f3a9861a63ce221b13d8bb
Successfully built nrclex
Installing collected packages: nrclex
Successfully installed nrclex-3.0.0
[nltk_data] Downloading package brown to /root/nltk_data...
[nltk_data]   Unzipping corpora/brown.zip.
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
[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 conll2000 to /root/nltk_data...
[nltk_data]   Unzipping corpora/conll2000.zip.
[nltk_data] Downloading package movie_reviews to /root/nltk_data...
[nltk_data]   Unzipping corpora/movie_reviews.zip.
```

```
# ==========================================
# CELL 2: LOAD & PREPARE DATASET
# ==========================================

try:
    print("Loading datasets...")
    df_d = pd.read_csv('d_tweets.csv')
    df_n = pd.read_csv('non_d_tweets.csv')

    # 1. Labeling
    df_d['label'] = 1
    df_n['label'] = 0

    # 2. Combining
    # We take only the 'tweet' column and the label
    df_d = df_d[['tweet', 'label']]
    df_n = df_n[['tweet', 'label']]

    df = pd.concat([df_d, df_n])

    # 3. Shuffling
    df = df.sample(frac=1, random_state=42).reset_index(drop=True)

    print(f"SUCCESS: Dataset Loaded.")
    print(f"Total Samples: {len(df)}")
    print(f"Depressed (1): {len(df[df['label']==1])}")
    print(f"Non-Depressed (0): {len(df[df['label']==0])}")

except FileNotFoundError:
    print("ERROR: Please upload 'd_tweets.csv' and 'non_d_tweets.csv' to Colab files.")
```

```
Loading datasets...
SUCCESS: Dataset Loaded.
Total Samples: 8305
Depressed (1): 3496
Non-Depressed (0): 4809
```

```
# ==========================================
# CELL 3: STRICT DATA CLEANING
# ==========================================
def preprocess_strict(text):
    # 1. Lowercase
    text = str(text).lower()

    # 2. Remove URLs, Handles, HTML
    text = re.sub(r'http\S+|www\S+|https\S+', '', text, flags=re.MULTILINE)
    text = re.sub(r'@\w+', '', text)
    text = re.sub(r'<.*?>', '', text)

    # 3. Remove Special Characters (Keep text only)
    text = re.sub(r'[^a-z\s]', '', text)

    # 4. NOTE: We deliberately KEEP stopwords ("I", "myself")
    # as per the paper's psycholinguistic requirement.
    return text.strip()
```

```
print("Cleaning data...")
df['clean_text'] = df['tweet'].apply(preprocess_strict)
# Remove empty rows after cleaning
df = df[df['clean_text'] != '']
print("Data Cleaning Complete.")
print(f"Sample: {df['clean_text'].iloc[0]}")
```

```
Cleaning data...
Data Cleaning Complete.
Sample: e and the smashbrosultimate invitational are
```

```
# ==========================================
# CELL 4: FEATURE ENGINEERING
# ==========================================
# A. Tokenization (LSTM Input)
tokenizer = Tokenizer(num_words=VOCAB_SIZE, oov_token="<OOV>")
tokenizer.fit_on_texts(df['clean_text'])
sequences = tokenizer.texts_to_sequences(df['clean_text'])
X_seq = pad_sequences(sequences, maxlen=SEQ_LENGTH, padding='post', truncating='post')

# B. NRC Emotion Vectors (Auxiliary Input)
def get_nrc_vector(text):
    emo = NRCLex(text)
    scores = emo.raw_emotion_scores
    # 8 Basic Emotions
    order = ['anger', 'fear', 'anticipation', 'trust', 'surprise', 'sadness', 'joy', 'disgust']
    vector = []
    # Normalize by text length
    length = len(text.split()) if len(text.split()) > 0 else 1
    for emotion in order:
        vector.append(scores.get(emotion, 0) / length)
    return np.array(vector)

print("Generating Emotion Vectors (This takes 1-2 mins)...")
# Using a loop for safety
X_emo = []
for t in df['clean_text']:
    X_emo.append(get_nrc_vector(t))
X_emo = np.array(X_emo)

y = df['label'].values
print(f"Features Ready. Sequence Shape: {X_seq.shape}, Emotion Shape: {X_emo.shape}")
```

```
Generating Emotion Vectors (This takes 1-2 mins)...
Features Ready. Sequence Shape: (8233, 100), Emotion Shape: (8233, 8)
```

```
# ==========================================
# CELL 5: FEDERATED PARTITIONING
# ==========================================
# 1. Held-out Test Set (20%)
X_seq_train, X_seq_test, X_emo_train, X_emo_test, y_train, y_test = train_test_split(
    X_seq, X_emo, y, test_size=0.2, random_state=42, stratify=y
)

# 2. Partition Training Data among K Clients
shard_size = len(y_train) // K_CLIENTS
clients = []
for i in range(K_CLIENTS):
    start = i * shard_size
    end = (i + 1) * shard_size
    clients.append({
        'seq': X_seq_train[start:end],
        'emo': X_emo_train[start:end],
        'y': y_train[start:end],
        'n_k': shard_size
    })

print(f"Data partitioned into {K_CLIENTS} Federated Clients.")
```

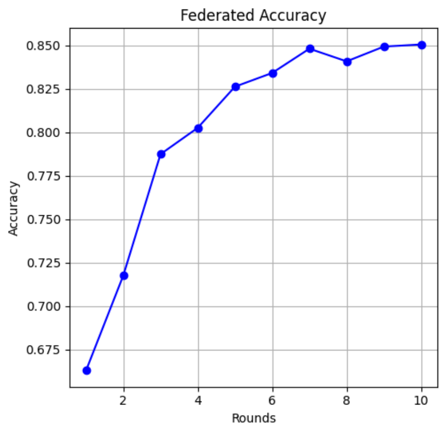
```
Data partitioned into 10 Federated Clients.
```

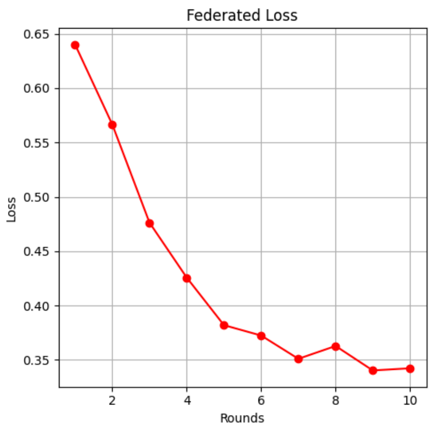
```
# ========================================
# CELL 6: FEDERATED LEARNING (FedAvg)
# ========================================
def build_model():
    # Input 1: Sequence
    in_seq = Input(shape=(SEQ_LENGTH,))
    emb = Embedding(VOCAB_SIZE, EMBED_DIM)(in_seq)
    bilstm = Bidirectional(LSTM(HIDDEN_DIM))(emb)

    # Input 2: Emotion Vector
    in_emo = Input(shape=(8,))
    emo_dense = Dense(16, activation='relu')(in_emo)

    # Fusion
    merged = Concatenate()([bilstm, emo_dense])
    drop = Dropout(0.5)(merged)
    out = Dense(1, activation='sigmoid')(drop)

    model = Model(inputs=[in_seq, in_emo], outputs=out)
    model.compile(optimizer=Adam(0.001), loss=BinaryCrossentropy(), metrics=['accuracy'])
    return model

# Initialize Global Model
global_model = build_model()
global_weights = global_model.get_weights()
history = {'acc': [], 'loss': []}

print(f"Starting Federated Training ({GLOBAL_ROUNDS} Rounds)...")

for r in range(GLOBAL_ROUNDS):
    scaled_local_weights = []

    # Simulate Clients
    for k in range(K_CLIENTS):
        local_model = build_model()
        local_model.set_weights(global_weights)

        # Local Training
        c = clients[k]
        local_model.fit([c['seq'], c['emo']], c['y'],
                        epochs=LOCAL_EPOCHS,
                        batch_size=BATCH_SIZE,
                        verbose=0)

        # Weight Scaling (FedAvg)
        w_k = local_model.get_weights()
        factor = c['n_k'] / len(y_train)
        scaled_w_k = [layer * factor for layer in w_k]
        scaled_local_weights.append(scaled_w_k)

    # Aggregation
    new_weights = [tf.math.reduce_sum(weights, axis=0) for weights in zip(*scaled_local_weights)]
    global_weights = new_weights
    global_model.set_weights(global_weights)

    # Global Evaluation
    loss, acc = global_model.evaluate([X_seq_test, X_emo_test], y_test, verbose=0)
    history['acc'].append(acc)
    history['loss'].append(loss)
    print(f"  Round {r+1}: Global Accuracy = {acc:.4f}")
```

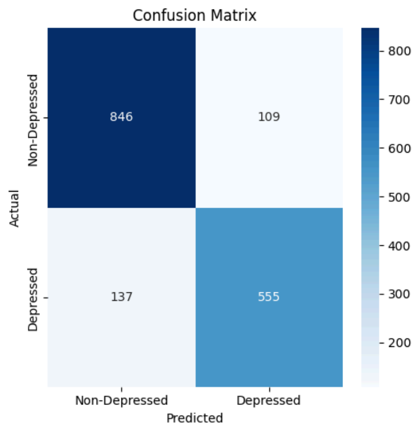
```
Starting Federated Training (10 Rounds)...
  Round 1: Global Accuracy = 0.6630
  Round 2: Global Accuracy = 0.7177
  Round 3: Global Accuracy = 0.7875
  Round 4: Global Accuracy = 0.8027
  Round 5: Global Accuracy = 0.8264
  Round 6: Global Accuracy = 0.8342
  Round 7: Global Accuracy = 0.8482
  Round 8: Global Accuracy = 0.8409
  Round 9: Global Accuracy = 0.8494
  Round 10: Global Accuracy = 0.8506
```

```
# ========================================
# CELL 7: RESULTS & VISUALIZATION
# ========================================
# 1. Metrics
preds_prob = global_model.predict([X_seq_test, X_emo_test])
preds = (preds_prob > 0.5).astype(int).flatten()

acc = accuracy_score(y_test, preds)
prec = precision_score(y_test, preds)
rec = recall_score(y_test, preds)
f1 = f1_score(y_test, preds)

print("\n=== FINAL RESEARCH RESULTS ===")
print(f"Accuracy:  {acc:.4f}")
print(f"Precision: {prec:.4f}")
print(f"Recall:    {rec:.4f}")
print(f"F1-Score:  {f1:.4f}")

# 2. Diagrams
plt.figure(figsize=(15, 5))

# Accuracy
plt.subplot(1, 3, 1)
plt.plot(range(1, GLOBAL_ROUNDS+1), history['acc'], 'b-o')
plt.title('Federated Accuracy')
plt.xlabel('Rounds')
plt.ylabel('Accuracy')
plt.grid(True)

# Loss
plt.subplot(1, 3, 2)
plt.plot(range(1, GLOBAL_ROUNDS+1), history['loss'], 'r-o')
plt.title('Federated Loss')
plt.xlabel('Rounds')
plt.ylabel('Loss')
plt.grid(True)

# Confusion Matrix
plt.subplot(1, 3, 3)
cm = confusion_matrix(y_test, preds)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Non-Depressed', 'Depressed'],
            yticklabels=['Non-Depressed', 'Depressed'])
plt.title('Confusion Matrix')
plt.ylabel('Actual')
plt.xlabel('Predicted')

plt.tight_layout()
plt.show()

print("\nDetailed Report:")
print(classification_report(y_test, preds, target_names=['Non-Depressed', 'Depressed']))
```

```
52/52 ━━━━━━━━━━━━━━━━━━━━ 1s 20ms/step

=== FINAL RESEARCH RESULTS ===
Accuracy:  0.8506
Precision: 0.8358
Recall:    0.8020
F1-Score:  0.8186
```







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

Non-Depressed       0.86      0.89      0.87       955
    Depressed       0.84      0.80      0.82       692

     accuracy                           0.85      1647
    macro avg       0.85      0.84      0.85      1647
 weighted avg       0.85      0.85      0.85      1647
```