```
!pip install kagglehub
import kagglehub

print("Downloading FER2013…")
path = kagglehub.dataset_download("msambare/fer2013")
print("Dataset downloaded at:", path)
```

```
Requirement already satisfied: kagglehub in /usr/local/lib/python3.12/dist-packages (0.3.13)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from kagglehub) (25.0)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.12/dist-packages (from kagglehub) (6.0.3)
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from kagglehub) (2.32.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from kagglehub) (4.67.1)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests->kaggleh
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests->kagglehub) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->kagglehub) (2
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests->kagglehub) (2
Downloading FER2013…
Using Colab cache for faster access to the 'fer2013' dataset.
Dataset downloaded at: /kaggle/input/fer2013
```

```
import os
import cv2
import numpy as np
from tensorflow.keras.preprocessing.image import img_to_array
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical

IMG_SIZE = 48
emotion_labels = ['angry', 'disgust', 'fear', 'happy', 'sad', 'surprise', 'neutral']
NUM_CLASSES = len(emotion_labels)

train_path = os.path.join(path, "train")

data = []
labels = []

print("Loading dataset…")

for idx, emotion in enumerate(emotion_labels):
    folder = os.path.join(train_path, emotion)
    for img_file in os.listdir(folder):
        img_path = os.path.join(folder, img_file)
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)

        if img is not None:
            img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
            img = img_to_array(img)
            data.append(img)
            labels.append(idx)

data = np.array(data, dtype="float32") / 255.0
data = data.reshape(-1, IMG_SIZE, IMG_SIZE, 1)
labels = to_categorical(np.array(labels), NUM_CLASSES)

X_train, X_val, y_train, y_val = train_test_split(data, labels, test_size=0.2)

print("Dataset ready.")
```

```
Loading dataset…
Dataset ready.
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization

def create_model():
    model = Sequential([
        Conv2D(32, (3,3), activation='relu', padding='same', input_shape=(48,48,1)),
        BatchNormalization(),
        Conv2D(32, (3,3), activation='relu', padding='same'),
        BatchNormalization(),
        MaxPooling2D(2,2),
        Dropout(0.25),

        Conv2D(64, (3,3), activation='relu', padding='same'),
        BatchNormalization(),
        Conv2D(64, (3,3), activation='relu', padding='same'),
        BatchNormalization(),
```

```
        MaxPooling2D(2,2),
        Dropout(0.25),

        Flatten(),
        Dense(256, activation='relu'),
        Dropout(0.5),
        Dense(NUM_CLASSES, activation='softmax')
    ])
    return model
```

```
# STEP 2: Improved Chatbot
def chatbot_start(emotion):
    responses = {
        "happy": "I'm happy to see you smiling today!",
        "sad": "I'm here with you. It’s okay to feel sad.",
        "angry": "Take a deep breath… I’m here for you.",
        "neutral": "How are you feeling right now?",
        "fear": "It’s okay, you’re safe now.",
        "surprise": "Oh! That seemed unexpected.",
    }
    return responses.get(emotion, "I'm here with you.")
```

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

model = create_model()
model.compile(optimizer=Adam(0.001), loss='categorical_crossentropy', metrics=['accuracy'])

checkpoint = ModelCheckpoint("best_model.keras", save_best_only=True, monitor="val_accuracy")
early = EarlyStopping(patience=5, restore_best_weights=True)

history = model.fit(
    X_train, y_train,
    validation_data=(X_val, y_val),
    epochs=40,
    batch_size=64,
    callbacks=[checkpoint, early]
)
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarning: Do not pass an `inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 24s 36ms/step - accuracy: 0.2428 - loss: 2.6015 - val_accuracy: 0.2518 - val_loss: 1.9257
Epoch 2/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step - accuracy: 0.3019 - loss: 1.7085 - val_accuracy: 0.4013 - val_loss: 1.5630
Epoch 3/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 15ms/step - accuracy: 0.3496 - loss: 1.6154 - val_accuracy: 0.4072 - val_loss: 1.4991
Epoch 4/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 15ms/step - accuracy: 0.3729 - loss: 1.5643 - val_accuracy: 0.4201 - val_loss: 1.4687
Epoch 5/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step - accuracy: 0.3898 - loss: 1.5245 - val_accuracy: 0.4619 - val_loss: 1.4237
Epoch 6/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step - accuracy: 0.4119 - loss: 1.4875 - val_accuracy: 0.4434 - val_loss: 1.4428
Epoch 7/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step - accuracy: 0.4190 - loss: 1.4670 - val_accuracy: 0.4742 - val_loss: 1.3786
Epoch 8/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 15ms/step - accuracy: 0.4219 - loss: 1.4503 - val_accuracy: 0.4904 - val_loss: 1.3488
Epoch 9/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step - accuracy: 0.4248 - loss: 1.4415 - val_accuracy: 0.4695 - val_loss: 1.3794
Epoch 10/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step - accuracy: 0.4371 - loss: 1.4069 - val_accuracy: 0.4897 - val_loss: 1.3431
Epoch 11/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step - accuracy: 0.4343 - loss: 1.3977 - val_accuracy: 0.4887 - val_loss: 1.3207
Epoch 12/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step - accuracy: 0.4539 - loss: 1.3640 - val_accuracy: 0.5010 - val_loss: 1.3044
Epoch 13/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 15ms/step - accuracy: 0.4646 - loss: 1.3317 - val_accuracy: 0.5152 - val_loss: 1.3021
Epoch 14/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step - accuracy: 0.4803 - loss: 1.3104 - val_accuracy: 0.4969 - val_loss: 1.3520
Epoch 15/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 15ms/step - accuracy: 0.4906 - loss: 1.2915 - val_accuracy: 0.5228 - val_loss: 1.2734
Epoch 16/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step - accuracy: 0.5055 - loss: 1.2529 - val_accuracy: 0.5261 - val_loss: 1.2641
Epoch 17/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 15ms/step - accuracy: 0.5167 - loss: 1.2260 - val_accuracy: 0.5366 - val_loss: 1.2869
Epoch 18/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step - accuracy: 0.5215 - loss: 1.2097 - val_accuracy: 0.5313 - val_loss: 1.2435
Epoch 19/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step - accuracy: 0.5277 - loss: 1.1990 - val_accuracy: 0.5369 - val_loss: 1.2481
Epoch 20/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step - accuracy: 0.5419 - loss: 1.1696 - val_accuracy: 0.5301 - val_loss: 1.2753
```

```
Epoch 21/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 15ms/step - accuracy: 0.5525 - loss: 1.1302 - val_accuracy: 0.5462 - val_loss: 1.2380
Epoch 22/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step - accuracy: 0.5621 - loss: 1.1235 - val_accuracy: 0.5500 - val_loss: 1.2448
Epoch 23/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step - accuracy: 0.5609 - loss: 1.1047 - val_accuracy: 0.5333 - val_loss: 1.2928
Epoch 24/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step - accuracy: 0.5709 - loss: 1.0794 - val_accuracy: 0.5495 - val_loss: 1.2583
Epoch 25/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step - accuracy: 0.5797 - loss: 1.0673 - val_accuracy: 0.5622 - val_loss: 1.2953
Epoch 26/40
359/359 ━━━━━━━━━━━━━━━━━━━━ 5s 14ms/step - accuracy: 0.5891 - loss: 1.0562 - val_accuracy: 0.5540 - val_loss: 1.2495
```

```
model.save("/content/smart_emotion_model.keras")
print("Model saved successfully.")
model.save("/content/emotion_model.h5")
print("emotion_model.h5 saved!")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file f
Model saved successfully.
emotion_model.h5 saved!
```

```
from tensorflow.keras.models import load_model

model = load_model("/content/smart_emotion_model.keras")
print("Model loaded.")

def predict_emotion(img_path):
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    img = cv2.resize(img, (48,48))
    img = img.reshape(1,48,48,1)/255.0
    pred = model.predict(img)[0]
    return emotion_labels[np.argmax(pred)], float(np.max(pred))
```

```
Model loaded.
```

```
from textblob import TextBlob

def analyze_text(text):
    polarity = TextBlob(text).sentiment.polarity
    if polarity < -0.3:
        return "Negative", "High Risk"
    elif polarity < 0:
        return "Slight Negative", "Medium Risk"
    else:
        return "Positive", "Low Risk"

def chatbot_start(emotion):
    if emotion in ["sad", "fear", "angry"]:
        return "Hey… you seem upset. Want to talk about it?"
    elif emotion == "happy":
        return "You're smiling! What made you happy today?"
    else:
        return "How are you feeling right now?"

def final_report(e, conf, mood, risk):
    print("\n------ FINAL REPORT ------")
    print("Facial Emotion:", e)
    print("Confidence:", round(conf,2))
    print("Conversation Mood:", mood)
    print("Risk Level:", risk)
```

```
def mental_health_tips(emotion):
    tips = {
        "happy": "Keep doing what makes you feel good! Spread your positivity today.",
        "sad": "Try taking a short walk, listening to calming music, or writing your thoughts.",
        "fear": "Deep breathing can help. Inhale for 4 seconds… hold… exhale slowly.",
        "angry": "Try drinking water, step away from the situation, or talk to someone you trust.",
        "neutral": "Maybe take a small break or do something relaxing today.",
        "surprise": "Unexpected moments can be overwhelming — take a moment to breathe."
    }
    return tips.get(emotion, "Taking care of yourself is important.")
```

```
from google.colab.patches import cv2_imshow
import cv2
```

```
# SMALL SQUARE IMAGE DISPLAY
def show_image(path):
    import matplotlib.pyplot as plt
    import matplotlib.image as mpimg

    img = mpimg.imread(path)
    plt.figure(figsize=(4,4))       # Small square
    plt.imshow(img)
    plt.axis('off')
    plt.title("Uploaded Image", fontsize=12)
    plt.tight_layout()
    plt.show()
```

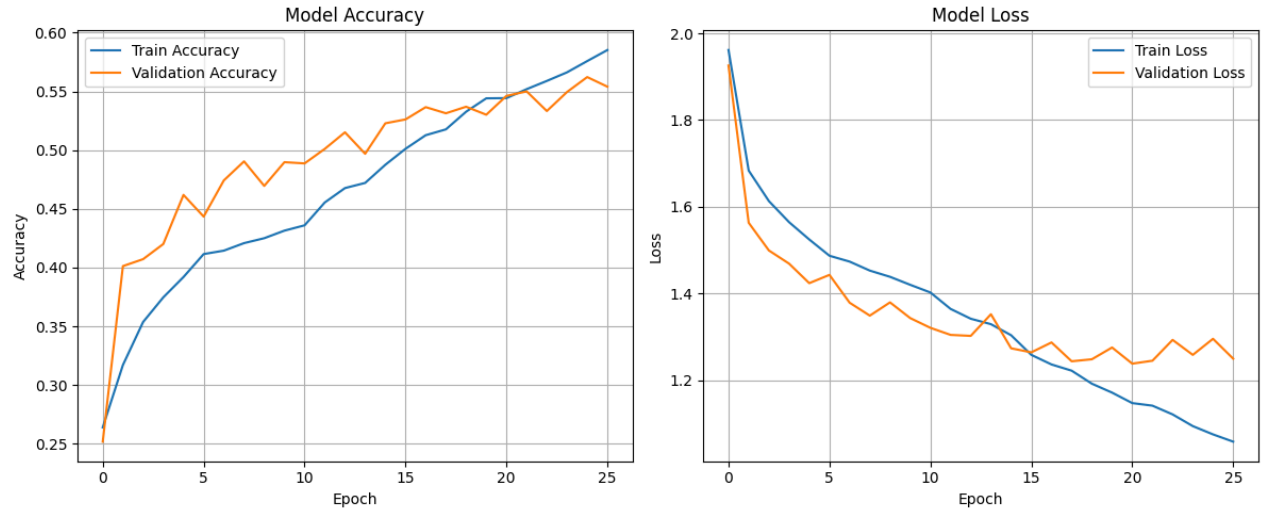
```
def final_report(emotion, confidence, mood, risk):
    print("\n------ CLINICAL ASSESSMENT REPORT ------")
    print(f"Observed Facial Emotion : {emotion.capitalize()}")
    print(f"Model Confidence        : {confidence:.2f}")
    print(f"Conversation Mood       : {mood}")
    print(f"Clinical Risk Estimate  : {risk}")

    print("\nDoctor’s Note:")

    if risk == "High Risk":
        print("Your responses indicate significant emotional distress. "
              "I strongly advise reaching out to a mental health professional immediately.")

    elif risk == "Medium Risk":
        print("There are noticeable signs of emotional discomfort. "
              "Try grounding exercises, and feel free to talk more about what’s bothering you.")

    else:
        print("Your responses appear stable. Still, I'm here if you’d like to discuss anything.")

    print("\nSuggested Wellness Practice:")
    print("• Deep Breathing (4-2-4 method): Inhale for 4 seconds… hold for 2… exhale for 4.")
    print("• Relax your shoulders and take a slow breath.")
    print("-----------------------------------------------\n")
```

```
def doctor_response(emotion):
    responses = {
        "happy": "Doctor: I’m glad to see positivity in your expression today. Keep nurturing it.",
        "sad": "Doctor: I can sense sadness. Thank you for sharing—I'm here to support you.",
        "fear": "Doctor: Your expression suggests fear or distress. You're safe now, and we’ll work through this together
        "angry": "Doctor: I sense some frustration. Let’s explore what’s causing it.",
        "neutral": "Doctor: Your face seems neutral. How have you been feeling lately?",
        "surprise": "Doctor: That looks like surprise or shock. Did something unexpected happen?"
    }
    return responses.get(emotion, "Doctor: I'm here to assist you. Tell me more.")
```

```
import matplotlib.pyplot as plt

# --- Training History Plot ---
print("\nGenerating Training History Plot...")
plt.figure(figsize=(12, 5))

# Plot Accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.grid(True)

# Plot Loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.grid(True)
```

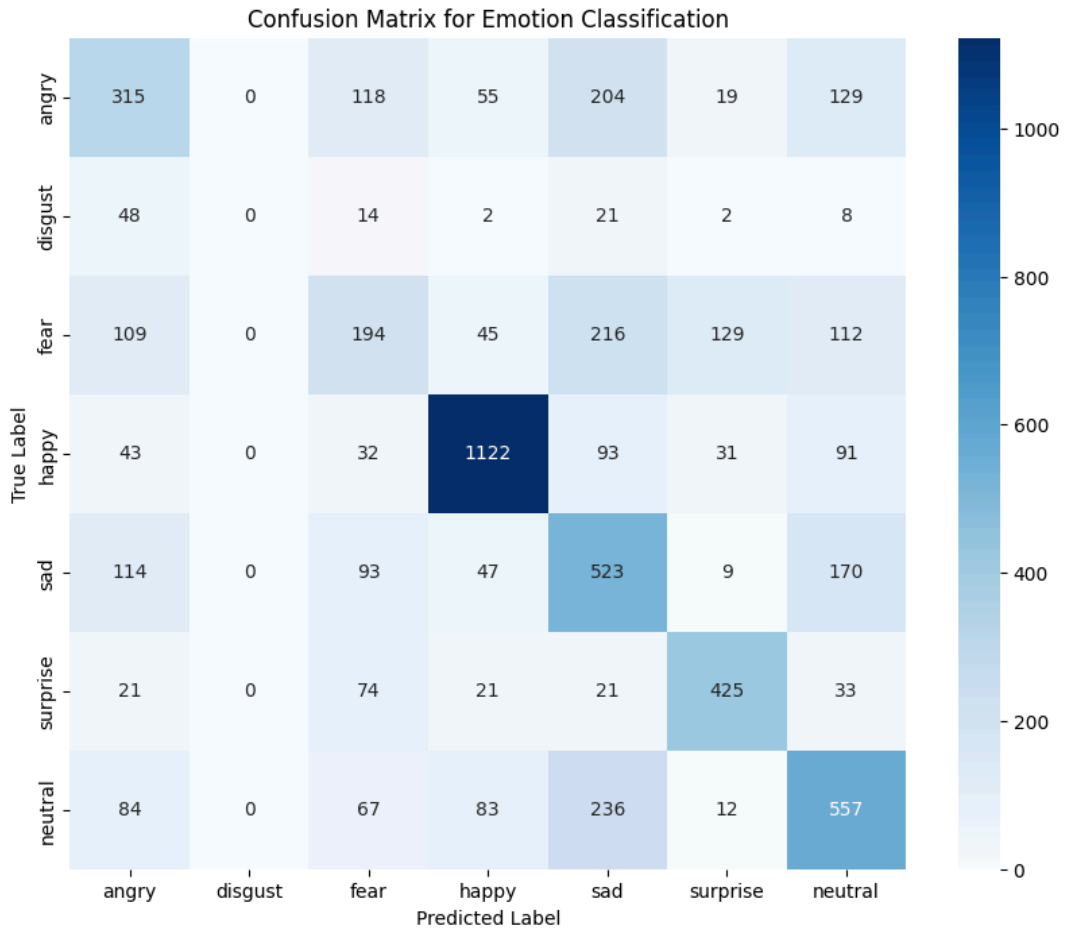
```
plt.tight_layout()
plt.show()
#
```

```
Generating Training History Plot...
```



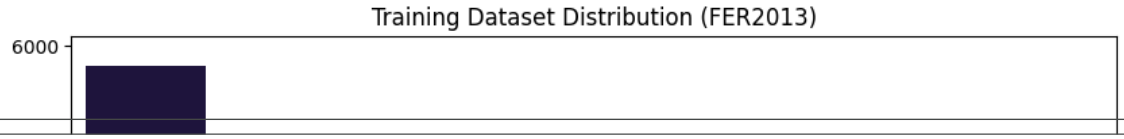

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
from numpy import argmax

# --- Confusion Matrix ---
print("\nGenerating Confusion Matrix...")
# 1. Get predictions on validation data
y_pred_probs = model.predict(X_val, verbose=0)
y_pred = argmax(y_pred_probs, axis=1)
y_true = argmax(y_val, axis=1)

# 2. Compute Confusion Matrix
cm = confusion_matrix(y_true, y_pred)

# 3. Plot
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=emotion_labels, yticklabels=emotion_labels)
plt.title('Confusion Matrix for Emotion Classification')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
plt.show()
```

```
Generating Confusion Matrix...
```




```
import pandas as pd

# --- Dataset Distribution Bar Chart (Train Data) ---
print("\nGenerating Dataset Distribution Bar Chart...")
# Reverse the one-hot encoding to get the original class indices
y_train_indices = np.argmax(y_train, axis=1)
emotion_counts = pd.Series(y_train_indices).map(lambda x: emotion_labels[x]).value_counts()

plt.figure(figsize=(10, 6))
sns.barplot(x=emotion_counts.index, y=emotion_counts.values, palette="magma")
plt.title('Training Dataset Distribution (FER2013)')
plt.xlabel('Emotion')
plt.ylabel('Number of Samples')
plt.show()
#
```

```
Generating Dataset Distribution Bar Chart...
/tmp/ipython-input-1654851856.py:10: FutureWarning:

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

  sns.barplot(x=emotion_counts.index, y=emotion_counts.values, palette="magma")
```




```
test_img = "/content/image.png"

show_image(test_img)

emotion, conf = predict_emotion(test_img)
print("Detected:", emotion, "| Confidence:", conf)

print("\n" + doctor_response(emotion))
u1 = input("You: ")

mood, risk = analyze_text(u1)
final_report(emotion, conf, mood, risk)
```


```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 600ms/step
Detected: sad | Confidence: 0.2508049011230469

Doctor: I can sense sadness. Thank you for sharing—I'm here to support you.
You:
```