Code:

```
 # Install required packages
!pip install hmmlearn
!pip install optuna

import pandas as pd
import numpy as np
import optuna
from sklearn.preprocessing import StandardScaler
from hmmlearn import hmm
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split, StratifiedKFold,cross_val_score
from sklearn.metrics import accuracy_score, classification_report,confusion_matrix,
ConfusionMatrixDisplay, f1_score
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

# Load data
df = pd.read_csv("cirrhosis_preprocessed (1).csv")

# Feature preparation
all_features = df.drop(columns=["Stage"]).columns.tolist()
X_raw = df[all_features]
y = df["Stage"].astype(int) - 1

# Scale and fit HMM
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_raw)
hmm_model = hmm.GaussianHMM(n_components=4, covariance_type="diag", n_iter
=100, random_state=42)
```

```
hmm_model.fit(X_scaled)
df["HMM_State"] = hmm_model.predict(X_scaled)
X = df[all_features + ["HMM_State"]]

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

# Optuna objective function
def objective(trial):
params = {
"n_estimators": trial.suggest_int("n_estimators", 50, 300),
"max_depth": trial.suggest_int("max_depth", 3, 10),
"learning_rate": trial.suggest_float("learning_rate", 0.01, 0.3),
"subsample": trial.suggest_float("subsample", 0.6, 1.0),
"colsample_bytree": trial.suggest_float("colsample_bytree", 0.6,1.0),
"gamma": trial.suggest_float("gamma", 0, 5),
"reg_alpha": trial.suggest_float("reg_alpha", 0, 1),
"reg_lambda": trial.suggest_float("reg_lambda", 0, 1),
"random_state": 42,
"use_label_encoder": False,
"eval_metric": "mlogloss"
}
model = XGBClassifier(**params)
cv = StratifiedKFold(n_splits=20, shuffle=True, random_state=42)
return cross_val_score(model, X_train, y_train, scoring="accuracy", cv=cv, n_jobs=-1).mean()

# Run Optuna
study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=30, timeout=600)

# Train final model
```

```
best_params = study.best_params
best_params.update({"use_label_encoder": False, "eval_metric": "mlogloss","random_state": 42})
final_model = XGBClassifier(**best_params)
final_model.fit(X_train, y_train)
y_proba = final_model.predict_proba(X_test)

# Thresholding function
def apply_custom_thresholds(proba, thresholds):
return np.argmax(proba / thresholds, axis=1)

# Threshold optimization
def threshold_objective(trial):
thresholds = [
trial.suggest_float("th_0", 0.05, 1.0),
trial.suggest_float("th_1", 0.05, 1.0),
trial.suggest_float("th_2", 0.05, 1.0),
trial.suggest_float("th_3", 0.05, 1.0),

y_pred = apply_custom_thresholds(y_proba, thresholds)
return f1_score(y_test, y_pred, average="macro")

threshold_study = optuna.create_study(direction="maximize")
threshold_study.optimize(threshold_objective, n_trials=50, timeout=300)

# Final predictions and evaluation
best_thresholds = [
threshold_study.best_params["th_0"],
threshold_study.best_params["th_1"],
threshold_study.best_params["th_2"],
threshold_study.best_params["th_3"]
```

```
y_pred_final = apply_custom_thresholds(y_proba, best_thresholds)


print("Final Accuracy:", accuracy_score(y_test, y_pred_final))
print("Classification Report:\n", classification_report(y_test,y_pred_final))


# Confusion matrix
cm = confusion_matrix(y_test, y_pred_final)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[0, 1, 2,3])
disp.plot(cmap="Blues", values_format="d")
plt.title("Confusion Matrix (XGBoost + HMM + Optimized Thresholds)")
plt.tight_layout()
plt.show()
```