```
!pip install xgboost
!pip install optuna
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.12/dist-packages (3.0.4)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from xgboost) (2
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.12/dist-packages (from x
Requirement already satisfied: scipy in /usr/local/lib/python3.12/dist-packages (from xgboost) (1
Collecting optuna
  Downloading optuna-4.5.0-py3-none-any.whl.metadata (17 kB)
Collecting alembic>=1.5.0 (from optuna)
  Downloading alembic-1.16.5-py3-none-any.whl.metadata (7.3 kB)
Collecting colorlog (from optuna)
  Downloading colorlog-6.9.0-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from optuna) (2.(
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from op
Requirement already satisfied: sqlalchemy>=1.4.2 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from optuna) (4.6
Requirement already satisfied: PyYAML in /usr/local/lib/python3.12/dist-packages (from optuna) (6
Requirement already satisfied: Mako in /usr/lib/python3/dist-packages (from alembic>=1.5.0->optuna
Requirement already satisfied: typing-extensions>=4.12 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: greenlet>=1 in /usr/local/lib/python3.12/dist-packages (from sqlal
Downloading optuna-4.5.0-py3-none-any.whl (400 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 400.9/400.9 kB 8.6 MB/s eta 0:00:00
Downloading alembic-1.16.5-py3-none-any.whl (247 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 247.4/247.4 kB 18.4 MB/s eta 0:00:00
Downloading colorlog-6.9.0-py3-none-any.whl (11 kB)
Installing collected packages: colorlog, alembic, optuna
Successfully installed alembic-1.16.5 colorlog-6.9.0 optuna-4.5.0
```

```
import pandas as pd
import numpy as np
import xgboost as xgb
import optuna
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.feature_selection import RFECV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
```

```
##load and prepare the data

def load_data(file_path):
    """Loads and prepares the depression dataset."""
    df = pd.read_csv('/content/Depression among female undergraduate students.csv')

    # Define target and features
    y = df['DepressionDiagnosed'].apply(lambda x: 1 if x == 'Yes' else 0)
    X = df.drop('DepressionDiagnosed', axis=1)

    # Identify categorical and numerical features
    categorical_features = X.select_dtypes(include=['object', 'category']).columns
    numerical_features = X.select_dtypes(include=np.number).columns

    return X, y, numerical_features, categorical_features



## data visualize
df = pd.read_csv('/content/Depression among female undergraduate students.csv')
display(df.head())
```

	Unnamed: 0	Timestamp	Continue	Age	Sex	Ethnicity	Yearinschool	Heght	Weight	ObesityHist
0	0	2019-08-26	Yes	21-24	Female	Latino	Senior	5.7	230.0	No
1	1	2019-08-26	Yes	18-21	Female	White	Junior	5.4	160.0	Yes
2	2	2019-08-26	Yes	21-24	Female	White	Senior	5.8	240.0	Yes
3	3	2019-08-26	Yes	18-21	Female	White	Freshman	5.9	155.0	No
4	4	2019-08-26	Yes	18-21	Female	White	Freshman	5.1	137.0	No

5 rows × 69 columns

```
##data preprocessing
def create_preprocessor(numerical_features, categorical_features):
    """Creates a scikit-learn pipeline for data pre-processing."""
    numerical_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='median')), # Median imputation
        ('scaler', MinMaxScaler()) # Min-Max normalization
    ])

    categorical_transformer = Pipeline(steps=[
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('onehot', OneHotEncoder(handle_unknown='ignore'))
    ])

    preprocessor = ColumnTransformer(
        transformers=[
            ('num', numerical_transformer, numerical_features),
            ('cat', categorical_transformer, categorical_features)
        ])
    return preprocessor


## define the objective function for hyperparameter tuning
def objective(trial, X, y):
    """Optuna objective function to find the best XGBoost hyperparameters."""
    params = {
        'objective': 'binary:logistic',
        'eval_metric': 'logloss',
        'n_estimators': trial.suggest_int('n_estimators', 100, 1000),
        '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', 0.5, 1.0),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.5, 1.0),
        'gamma': trial.suggest_float('gamma', 0, 5),
        'reg_alpha': trial.suggest_float('reg_alpha', 0, 5),
        'reg_lambda': trial.suggest_float('reg_lambda', 0, 5),
        'use_label_encoder': False
    }


## define the objective function for hyperparameter tuning
def objective(trial, X, y):
    """Optuna objective function to find the best XGBoost hyperparameters."""
    params = {
        'objective': 'binary:logistic',
        'eval_metric': 'logloss',
        'n_estimators': trial.suggest_int('n_estimators', 100, 1000),
        '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', 0.5, 1.0),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.5, 1.0),
        'gamma': trial.suggest_float('gamma', 0, 5),
        'reg_alpha': trial.suggest_float('reg_alpha', 0, 5),
        'reg_lambda': trial.suggest_float('reg_lambda', 0, 5),
        'use_label_encoder': False
    }

    # Use class weighting for imbalanced data
    scale_pos_weight = (y == 0).sum() / (y == 1).sum()
    params['scale_pos_weight'] = scale_pos_weight

    model = xgb.XGBClassifier(**params)

    # Perform cross-validation and optimize for F1-score
    cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)
    scores = []
    for train_idx, val_idx in cv.split(X, y):
        X_train, X_val = X[train_idx], X[val_idx]
        y_train, y_val = y[train_idx], y[val_idx]


        model.fit(X_train, y_train)
        preds = model.predict(X_val)
        scores.append(f1_score(y_val, preds))

    return np.mean(scores)

# Main script execution
if __name__ == '__main__':
    # Load data
    file_path = '/content/Depression among female undergraduate students.csv'
    X, y, numerical_features, categorical_features = load_data(file_path)

    # Create pre-processor
    preprocessor = create_preprocessor(numerical_features, categorical_features)

    # Define the outer cross-validation strategy
    N_SPLITS = 5
    skf = StratifiedKFold(n_splits=N_SPLITS, shuffle=True, random_state=42)

    metrics = {'accuracy': [], 'precision': [], 'recall': [], 'f1': [], 'auc': []}

    print(f"Starting {N_SPLITS}-fold cross-validation...")

    for fold, (train_idx, test_idx) in enumerate(skf.split(X, y)):
        print(f"--- Fold {fold+1}/{N_SPLITS} ---")

        # Split data for this fold
        X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
        y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

        # 2.1 Pre-processing
        # Fit pre-processor on training data and transform both sets
        X_train_prep = preprocessor.fit_transform(X_train)
        X_test_prep = preprocessor.transform(X_test)

        # 2.2 Feature Selection (ECO Alternative)
        print("Running feature selection with RFECV...")
        xgb_fs = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss')
        rfecv = RFECV(
            estimator=xgb_fs,
            step=1,
            cv=StratifiedKFold(3),
            scoring='f1',
            min_features_to_select=5 # Ensure a minimum number of features
        )
        rfecv.fit(X_train_prep, y_train)

        X_train_fs = rfecv.transform(X_train_prep)
        X_test_fs = rfecv.transform(X_test_prep)
        print(f"Selected {X_train_fs.shape[1]} features out of {X_train_prep.shape[1]}.")

        # 2.4 Hyper-parameter Tuning (WSO Alternative)
        print("Running hyper-parameter tuning with Optuna...")
        study = optuna.create_study(direction='maximize')
        study.optimize(lambda trial: objective(trial, X_train_fs, y_train.values), n_trials=50) # Pas

        best_params = study.best_params
        print(f"Best F1-score from tuning: {study.best_value:.4f}")

        # Train final model with best parameters
        final_model = xgb.XGBClassifier(**best_params, use_label_encoder=False)
        final_model.fit(X_train_fs, y_train)

        # Evaluate on the held-out test fold
        y_pred = final_model.predict(X_test_fs)
        y_pred_proba = final_model.predict_proba(X_test_fs)[:, 1]

        # Collect metrics
        metrics['accuracy'].append(accuracy_score(y_test, y_pred))
        metrics['precision'].append(precision_score(y_test, y_pred))
        metrics['recall'].append(recall_score(y_test, y_pred))
        metrics['f1'].append(f1_score(y_test, y_pred))
        metrics['auc'].append(roc_auc_score(y_test, y_pred_proba))

    # Print average results across all folds
    print("\n--- Final Results (Averaged Across Folds) ---")
    for key, value in metrics.items():
        print(f"{key.capitalize():<10}: {np.mean(value):.4f} \u00B1 {np.std(value):.4f}")
```

```
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning: [06:18:52] WARNING
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
Best F1-score from tuning: 0.9354

--- Final Results (Averaged Across Folds) ---
Accuracy  : 0.9493 ± 0.0139
Precision : 0.9931 ± 0.0138
Recall    : 0.8727 ± 0.0297
F1        : 0.9288 ± 0.0200
Auc       : 0.9384 ± 0.0094
```

Double-click (or enter) to edit