```
import pandas as pd
import numpy as np
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from sklearn.metrics import accuracy_score
```

```
train_fd=pd.read_csv("/content/sample_data/train.csv")
test_fd=pd.read_csv("/content/sample_data/test.csv")
```

```
train_fd.head()
```

	id	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	
0	0	Male	24.443011	1.699998	81.669950	yes	yes	2.000000	2.983297	Sometimes	no	2.7
1	1	Female	18.000000	1.560000	57.000000	yes	yes	2.000000	3.000000	Frequently	no	2.0
2	2	Female	18.000000	1.711460	50.165754	yes	yes	1.880534	1.411685	Sometimes	no	1.9
3	3	Female	20.952737	1.710730	131.274851	yes	yes	3.000000	3.000000	Sometimes	no	1.6
4	4	Male	31.641081	1.914186	93.798055	yes	yes	2.679664	1.971472	Sometimes	no	1.9

Next steps: Generate code with `train_fd` New interactive sheet

```
test_fd.head()
```

	id	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE
0	15533	Female	19.007177	1.772449	137.852618	yes	yes	3.000000	3.000000	Sometimes	no
1	15534	Female	21.572114	1.698346	75.000000	yes	yes	2.000000	3.000000	Sometimes	no
2	15535	Male	22.285024	1.737453	82.000000	yes	yes	2.000000	2.720642	Sometimes	no
3	15536	Male	30.916426	1.775580	120.860386	yes	yes	2.712747	3.000000	Sometimes	no
4	15537	Female	18.000000	1.670000	65.000000	no	yes	2.000000	3.000000	Sometimes	no

Next steps: Generate code with `test_fd` New interactive sheet

```
X_train = train_fd.iloc[:, 1:17]
y_train = train_fd.iloc[:, 17]

X_test=test_fd.iloc[:,1:17]
```

```
categorical_cols = ['Gender', 'family_history_with_overweight', 'FAVC', 'CAEC',
                    'SMOKE', 'SCC', 'CALC', 'MTRANS']
```

```
from sklearn.preprocessing import LabelEncoder

target_encoder = LabelEncoder()
y_train = target_encoder.fit_transform(y_train)

le = LabelEncoder()
for col in categorical_cols:
    X_train[col] = le.fit_transform(X_train[col])
```

```
for col in categorical_cols:
    X_test[col] = le.fit_transform(X_test[col])
```

```
X_test
```

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O
0	0	19.007177	1.772449	137.852618	1	1	3.000000	3.000000	2	0	2.007348
1	0	21.572114	1.698346	75.000000	1	1	2.000000	3.000000	2	0	2.000000
2	1	22.285024	1.737453	82.000000	1	1	2.000000	2.720642	2	0	1.830614
3	1	30.916426	1.775580	120.860386	1	1	2.712747	3.000000	2	0	2.144368
4	0	18.000000	1.670000	65.000000	0	1	2.000000	3.000000	2	0	2.000000
...	...	...	...	...	...	...	...	...	...	...	...
5220	1	25.137087	1.766626	114.187096	1	1	2.919584	3.000000	2	0	2.151809
5221	1	18.000000	1.710000	50.000000	0	1	3.000000	4.000000	1	0	1.000000
5222	1	20.101026	1.819557	105.580491	1	1	2.407817	3.000000	2	0	2.000000
5223	1	33.852953	1.700000	83.520113	1	1	2.671238	1.971472	2	0	2.144838
5224	1	26.680376	1.816547	118.134898	1	1	3.000000	3.000000	2	0	2.003563

5225 rows × 16 columns

Next steps: Generate code with X_test New interactive sheet

```
X_train.head()
```

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SC
0	1	24.443011	1.699998	81.669950	1	1	2.000000	2.983297	2	0	2.763573	
1	0	18.000000	1.560000	57.000000	1	1	2.000000	3.000000	1	0	2.000000	
2	0	18.000000	1.711460	50.165754	1	1	1.880534	1.411685	2	0	1.910378	
3	0	20.952737	1.710730	131.274851	1	1	3.000000	3.000000	2	0	1.674061	
4	1	31.641081	1.914186	93.798055	1	1	2.679664	1.971472	2	0	1.979848	

Next steps: Generate code with X_train New interactive sheet

```
import numpy as np

num_classes = len(np.unique(y_train))
print("Number of classes:", num_classes)
```

```
Number of classes: 7
```

1.Random Forest 🌴

```
rf_random.fit(X_train, y_train)
print("Best Parameters:", rf_random.best_params_)
print("Best CV Accuracy:", rf_random.best_score_)
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits
Best Parameters: {'n_estimators': 600, 'min_samples_split': 10, 'min_samples_leaf': 1, 'max_features': 'log2', 'max_c
Best CV Accuracy: 0.8992476328818191
```

```
best_model = rf_random.best_estimator_
print("Best Hyperparameters:", rf_random.best_params_)
```

```
Best Hyperparameters: {'n_estimators': 600, 'min_samples_split': 10, 'min_samples_leaf': 1, 'max_features': 'log2', '
```

```
best_model.fit(
    X_train, y_train,
)
```

```
▾                       RandomForestClassifier                  ⓘ ?
RandomForestClassifier(bootstrap=False, max_features='log2',
                       min_samples_split=10, n_estimators=600, random_state=42)
```

```
y_test_bm = best_model.predict(X_test)
```

```
y_test_bm
```

```
array([4, 5, 6, ..., 2, 6, 3])
```

```
predicted_labels = target_encoder.inverse_transform(y_test_bm)
```

```
predicted_labels
```

```
array(['Obesity_Type_III', 'Overweight_Level_I', 'Overweight_Level_II',
       ..., 'Obesity_Type_I', 'Overweight_Level_II', 'Obesity_Type_II'],
      dtype=object)
```

```
output_df = pd.DataFrame({
    'id': test_fd['id'] if 'id' in test_fd.columns else range(1, len(test_fd)+1),
    'Predicted_WeightCategory': predicted_labels
})
```

```
output_df.to_csv('predictions_RF.csv', index=False)
print("Predictions saved to predictions.csv")
```

```
Predictions saved to predictions.csv
```

Random Forest gave us and accuracy of about 89.476% when checked with the test_y result in kaggle and 89.96% in final

2.AdaBoost 🌠

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
base_model = DecisionTreeClassifier(max_depth=1)
ada_model = AdaBoostClassifier(
    estimator=base_model,
    n_estimators=100,
    learning_rate=0.8,
    random_state=42
)
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import AdaBoostClassifier

ada = AdaBoostClassifier(random_state=42)

param_grid = {
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.01, 0.1, 0.5, 1.0]
}

grid = GridSearchCV(estimator=ada, param_grid=param_grid, cv=5, scoring='accuracy')
grid.fit(X_train, y_train)

print("Best Parameters:", grid.best_params_)
print("Best CV Accuracy:", grid.best_score_)

best_model = grid.best_estimator_
```

```
Best Parameters: {'learning_rate': 0.5, 'n_estimators': 150}
Best CV Accuracy: 0.7235569889647434
```

```
y_test_bm2 = best_model.predict(X_test)
```

```
predicted_labels2 = target_encoder.inverse_transform(y_test_bm2)
```

```
predicted_labels2
```

```
array(['Obesity_Type_III', 'Normal_Weight', 'Overweight_Level_II', ...,
       'Obesity_Type_I', 'Overweight_Level_II', 'Obesity_Type_III'],
      dtype=object)
```

```
output_df2 = pd.DataFrame({
    'id': test_fd['id'] if 'id' in test_fd.columns else range(1, len(test_fd)+1),
    'Predicted_WeightCategory': predicted_labels2
})
```

```
output_df.to_csv('predictions_ADA.csv', index=False)
print("Predictions saved to predictions_ADA.csv")
```

```
Predictions saved to predictions_ADA.csv
```

Ada Boost gave Accuracy of 87% in prediction in kaggle and 89% in final

3. XGBoost WO HyperTune 🎷 (Random Search_CV)

```
xgb = XGBClassifier(
    objective='multi:softprob',
    eval_metric='mlogloss',
    num_class=num_classes,
    use_label_encoder=False,
    random_state=42,
    n_jobs=-1
)
param_grid = {
    'n_estimators': [800, 1000, 1200, 1500],
    'learning_rate': [0.01, 0.02, 0.03, 0.05],
    'max_depth': [3, 4, 5, 6],
    'min_child_weight': [1, 2, 3, 5],
    'gamma': [0, 0.1, 0.3, 0.5],
    'subsample': [0.7, 0.8, 0.9],
    'colsample_bytree': [0.7, 0.8, 0.9],
    'reg_alpha': [0, 0.01, 0.1, 0.5],
    'reg_lambda': [1, 1.5, 2, 3]
}
random_search = RandomizedSearchCV(
    estimator=xgb,
    param_distributions=param_grid,
    n_iter=20,
    scoring='accuracy',
    cv=3,
    verbose=2,
    random_state=42,
    n_jobs=-1
)
```

```
random_search.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: UserWarning: [19:50:56] WARNING: /workspace/src/lear
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

▸ RandomizedSearchCV ⓘ ?

▸ best_estimator_: XGBClassifier

▸ XGBClassifier ?

```
best_model3 = random_search.best_estimator_
print("Best Hyperparameters:", random_search.best_params_)
```

```
Best Hyperparameters: {'subsample': 0.7, 'reg_lambda': 3, 'reg_alpha': 0.01, 'n_estimators': 800, 'min_child_weight':
```

```
best_model3.fit(
    X_train, y_train,
    verbose=True
)
```

```
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: UserWarning: [19:54:34] WARNING: /workspace/src/lear
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

```
▾                              XGBClassifier                              ⓘ ?

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=0.8, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='mlogloss',
              feature_types=None, feature_weights=None, gamma=0.1,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.02, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=6, max_leaves=None,
              min_child_weight=3, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=800, n_jobs=-1, num_class=7, ...)
```

```
y_test_bm = best_model3.predict(X_test)
```

```
y_test_bm
```

```
array([4, 5, 6, ..., 2, 6, 3])
```

```
predicted_labels = target_encoder.inverse_transform(y_test_bm)
```

```
predicted_labels
```

```
array(['Obesity_Type_III', 'Overweight_Level_I', 'Overweight_Level_II',
       ..., 'Obesity_Type_I', 'Overweight_Level_II', 'Obesity_Type_II'],
      dtype=object)
```

```
output_df = pd.DataFrame({
    'id': test_fd['id'] if 'id' in test_fd.columns else range(1, len(test_fd)+1),
    'Predicted_WeightCategory': predicted_labels
})
```

```
output_df.to_csv('predictions_XG_WO_Tuned.csv', index=False)
print("Predictions saved to predictions.csv")
```

```
Predictions saved to predictions.csv
```

Gave Accuracy of 91.1% in kaggle and 91.7% in final result

4.XG_Boost_Tuned 💯 (Optuna)

```
!pip install optuna
```

```
Collecting optuna
  Downloading optuna-4.5.0-py3-none-any.whl.metadata (17 kB)
Requirement already satisfied: alembic>=1.5.0 in /usr/local/lib/python3.12/dist-packages (from optuna) (1.17.0)
Collecting colorlog (from optuna)
  Downloading colorlog-6.10.1-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from optuna) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from optuna) (25.0)
Requirement already satisfied: sqlalchemy>=1.4.2 in /usr/local/lib/python3.12/dist-packages (from optuna) (2.0.44)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from optuna) (4.67.1)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.12/dist-packages (from optuna) (6.0.3)
Requirement already satisfied: Mako in /usr/local/lib/python3.12/dist-packages (from alembic>=1.5.0->optuna) (1.3.10)
Requirement already satisfied: typing-extensions>=4.12 in /usr/local/lib/python3.12/dist-packages (from alembic>=1.5.
Requirement already satisfied: greenlet>=1 in /usr/local/lib/python3.12/dist-packages (from sqlalchemy>=1.4.2->optuna
Requirement already satisfied: MarkupSafe>=0.9.2 in /usr/local/lib/python3.12/dist-packages (from Mako->alembic>=1.5.
Downloading optuna-4.5.0-py3-none-any.whl (400 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 400.9/400.9 kB 8.4 MB/s eta 0:00:00
Downloading colorlog-6.10.1-py3-none-any.whl (11 kB)
Installing collected packages: colorlog, optuna
Successfully installed colorlog-6.10.1 optuna-4.5.0
```

```python
import optuna
from xgboost import XGBClassifier
from sklearn.model_selection import cross_val_score
import numpy as np

def objective(trial):
    params = {
        'n_estimators': trial.suggest_int('n_estimators', 450, 550),
        'learning_rate': trial.suggest_float('learning_rate', 0.04, 0.07),
        'max_depth': trial.suggest_int('max_depth', 3, 5),
        'min_child_weight': trial.suggest_int('min_child_weight', 1, 3),
        'gamma': trial.suggest_float('gamma', 0.30, 0.36),
        'subsample': trial.suggest_float('subsample', 0.75, 0.82),
        'colsample_bytree': trial.suggest_float('colsample_bytree', 0.48, 0.54),
        'reg_alpha': trial.suggest_float('reg_alpha', 0.45, 0.60),
        'reg_lambda': trial.suggest_float('reg_lambda', 0.05, 0.15),

        # fixed values
        'objective': 'multi:softprob',
        'eval_metric': 'mlogloss',
        'num_class': num_classes,
        'random_state': 42,
        'n_jobs': -1
    }

    model = XGBClassifier(**params)
    scores = cross_val_score(model, X_train, y_train, cv=3, scoring='accuracy', n_jobs=-1)
    return np.mean(scores)

# Run the new Optuna study
study = optuna.create_study(direction='maximize', study_name="XGB_Optuna_FineTuned")
study.optimize(objective, n_trials=50, show_progress_bar=True)

# Show best results
print("Best Trial:")
trial = study.best_trial
print(f"  Accuracy: {trial.value}")
print("  Best Hyperparameters:")
for key, value in trial.params.items():
    print(f"    {key}: {value}")

# Train final model with tuned parameters
best_params = trial.params
best_xgb = XGBClassifier(
    **best_params,
    objective='multi:softprob',
    eval_metric='mlogloss',
    num_class=num_classes,
    use_label_encoder=False,
    random_state=42,
    n_jobs=-1
)
best_xgb.fit(X_train, y_train)
```