```
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import KFold
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder, LabelEncoder
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from sklearn.calibration import CalibratedClassifierCV
from imblearn.pipeline import make_pipeline
from imblearn.under_sampling import TomekLinks
from sklearn.metrics import auc, roc_curve
import joblib
import matplotlib.pyplot as plt
import seaborn as sns

def xgbc(df_x, df_y, based, tuned, calibrated):
    based_auc = []
    tuned_auc = []
    tuned_fi = []
    tprs = []
    calibrated_auc = []
    kf = KFold(n_splits=5, shuffle=True, random_state=19)
    for train_index, test_index in kf.split(df_x):
        #split
        df_x_train, df_x_test = df_x.iloc[train_index, :], df_x.iloc[test_index, :]
        df_y_train, df_y_test = df_y.iloc[train_index], df_y.iloc[test_index]
        #print(df_x_train, df_x_test, df_y_train, df_y_test)

        # continuous
        scaler = MinMaxScaler()
        df_x_train[continuous] = scaler.fit_transform(df_x_train[continuous])
        df_x_test[continuous] = scaler.transform(df_x_test[continuous])

        # nominal
        ohe = OneHotEncoder(sparse=False, handle_unknown = "ignore")
        nominal_train = ohe.fit_transform(df_x_train[nominal])
        nominal_name = ohe.get_feature_names(nominal)
        df_x_train = pd.concat([df_x_train, pd.DataFrame(nominal_train,
index=train_index, columns=nominal_name)], axis=1)
        df_x_train = df_x_train.drop(nominal, axis=1)
        nominal_test = ohe.transform(df_x_test[nominal])
        nominal_name = ohe.get_feature_names(nominal)
        df_x_test = pd.concat([df_x_test, pd.DataFrame(nominal_test, index=test_index,
columns=nominal_name)], axis=1)
        df_x_test = df_x_test.drop(nominal, axis=1)

        # label
        le = LabelEncoder()
        df_y_train = le.fit_transform(df_y_train)
        df_y_test = le.transform(df_y_test)

        # class weight
        df_y_train_1 = sum(df_y_train)
        df_y_train_0 = len(df_y_train) - sum(df_y_train)

        # based xgbc
        xgbc = XGBClassifier(booster="gbtree", random_state=19,
use_label_encoder=False, eval_metric="auc", tree_method="hist", n_jobs=-1)
        xgbc_based = xgbc.fit(df_x_train, df_y_train)
        joblib.dump(xgbc_based, based)
        df_y_test_pred_based = xgbc_based.predict_proba(df_x_test)
        fpr, tpr, thresholds = roc_curve(df_y_test, df_y_test_pred_based[:, 1])
        test_auroc_based = auc(fpr, tpr)
        based_auc.append(test_auroc_based)
```

```
        # tuned xgbc
        hyperparameters_xgbc = {"xgbclassifier__learning_rate": (0.01, 0.1, 1, 10),
                                "xgbclassifier__max_depth": (25, 75, 150),
                                "xgbclassifier__subsample": (0.2, 0.5, 0.8),
                                "xgbclassifier__colsample_bytree": (0.2, 0.5, 0.8),
                                "xgbclassifier__reg_lambda": (0.01, 0.1, 1, 10),
                                "xgbclassifier__reg_alpha": (0.01, 0.1, 1, 10),
                                "xgbclassifier__gamma": (0.01, 0.1, 1, 10),
                                "xgbclassifier__n_estimators": (25, 75, 150),
                                "xgbclassifier__scale_pos_weight": (df_y_train_1 /
df_y_train_0, df_y_train_0 / df_y_train_1)}

        pipeline = make_pipeline(TomekLinks(), XGBClassifier(booster="gbtree",
random_state=19, use_label_encoder=False, eval_metric="auc", tree_method="hist"))

        xgbc_rscv = RandomizedSearchCV(estimator=pipeline,
                                       param_distributions=hyperparameters_xgbc,
                                       n_jobs=-1,
                                       scoring="roc_auc",
                                       verbose=5,
                                       cv=5,
                                       n_iter=500,
                                       random_state=19)
        xgbc_tuned = xgbc_rscv.fit(df_x_train, df_y_train)
        joblib.dump(xgbc_tuned, tuned)
        df_y_test_pred_tuned = xgbc_tuned.predict_proba(df_x_test)
        fpr, tpr, thresholds = roc_curve(df_y_test, df_y_test_pred_tuned[:, 1])
        test_auroc_tuned = auc(fpr, tpr)
        tuned_auc.append(test_auroc_tuned)

        interp_tpr = np.interp(mean_fpr, fpr, tpr)
        interp_tpr[0] = 0.0
        tprs.append(interp_tpr)

        xgbc_tuned_fi =
xgbc_tuned.best_estimator_._final_estimator.feature_importances_
        xgbc_tuned_fi_df = pd.DataFrame(xgbc_tuned_fi)
        xgbc_tuned_fi_df.index = df_x_train.columns
        xgbc_tuned_fi_df.columns = (["Value"])
        tuned_fi.append(xgbc_tuned_fi_df)

        # calibrated xgbc
        xgbc_cccv = CalibratedClassifierCV(base_estimator=xgbc_tuned.best_estimator_,
cv=5)
        xgbc_calibrated = xgbc_cccv.fit(df_x_train, df_y_train)
        joblib.dump(xgbc_calibrated, calibrated)
        df_y_test_pred_calibrated = xgbc_calibrated.predict_proba(df_x_test)
        fpr, tpr, thresholds = roc_curve(df_y_test, df_y_test_pred_calibrated[:, 1])
        test_auroc_calibrated = auc(fpr, tpr)
        calibrated_auc.append(test_auroc_calibrated)

    return based_auc, tuned_auc, calibrated_auc, tprs, tuned_fi

def logistic(df_x, df_y, based):
    based_auc = []
    tprs = []
    kf = KFold(n_splits=5, shuffle=True, random_state=19)
    for train_index, test_index in kf.split(df_x):
        #split
        df_x_train, df_x_test = df_x.iloc[train_index, :], df_x.iloc[test_index, :]
        df_y_train, df_y_test = df_y.iloc[train_index], df_y.iloc[test_index]
        #print(df_x_train, df_x_test, df_y_train, df_y_test)

        # continuous
        scaler = MinMaxScaler()
        df_x_train[continuous] = scaler.fit_transform(df_x_train[continuous])
        df_x_test[continuous] = scaler.transform(df_x_test[continuous])
```

```
        # nominal
        ohe = OneHotEncoder(sparse=False, handle_unknown = "ignore")
        nominal_train = ohe.fit_transform(df_x_train[nominal])
        nominal_name = ohe.get_feature_names(nominal)
        df_x_train = pd.concat([df_x_train, pd.DataFrame(nominal_train,
index=train_index, columns=nominal_name)], axis=1)
        df_x_train = df_x_train.drop(nominal, axis=1)
        nominal_test = ohe.transform(df_x_test[nominal])
        nominal_name = ohe.get_feature_names(nominal)
        df_x_test = pd.concat([df_x_test, pd.DataFrame(nominal_test, index=test_index,
columns=nominal_name)], axis=1)
        df_x_test = df_x_test.drop(nominal, axis=1)

        # label
        le = LabelEncoder()
        df_y_train = le.fit_transform(df_y_train)
        df_y_test = le.transform(df_y_test)

        # class weight
        # df_y_train_1 = sum(df_y_train)
        # df_y_train_0 = len(df_y_train) - sum(df_y_train)

        # based lr
        lr = LogisticRegression(random_state=19, n_jobs=-1)
        lr_based = lr.fit(df_x_train, df_y_train)
        joblib.dump(lr_based, based)
        df_y_test_pred_based = lr_based.predict_proba(df_x_test)
        fpr, tpr, thresholds = roc_curve(df_y_test, df_y_test_pred_based[:, 1])
        test_auroc_based = auc(fpr, tpr)
        based_auc.append(test_auroc_based)

        interp_tpr = np.interp(mean_fpr, fpr, tpr)
        interp_tpr[0] = 0.0
        tprs.append(interp_tpr)

    return based_auc, tprs

def roc_ccurve(tuned_auc, tprs, based_auc_lr, tprs_lr, name, save_name):
    mean_tpr = np.mean(tprs, axis=0)
    mean_tpr[-1] = 1.0
    mean_auc = auc(mean_fpr, mean_tpr)
    std_auc = np.std(tuned_auc)
    plt.plot(mean_fpr, mean_tpr, color="b", label=r"XGboost (AUROC = %0.3f $\pm$
%0.3f)" % (mean_auc, std_auc), lw=2, alpha=0.8)

    std_tpr = np.std(tprs, axis=0)
    tprs_upper = np.minimum(mean_tpr + std_tpr, 1)
    tprs_lower = np.maximum(mean_tpr - std_tpr, 0)
    #plt.fill_between(mean_fpr, tprs_lower, tprs_upper, color="grey", alpha=0.2,
label=r"$\pm$ 1 std")
    plt.fill_between(mean_fpr, tprs_lower, tprs_upper, color="b", alpha=0.2)

    mean_tpr_lr = np.mean(tprs_lr, axis=0)
    mean_tpr_lr[-1] = 1.0
    mean_auc_lr = auc(mean_fpr, mean_tpr_lr)
    std_auc_lr = np.std(based_auc_lr)
    plt.plot(mean_fpr, mean_tpr_lr, color="r", label=r"Logistic Regression (AUROC =
%0.3f $\pm$ %0.3f)" % (mean_auc_lr, std_auc_lr), lw=2, alpha=0.8)

    std_tpr_lr = np.std(tprs_lr, axis=0)
    tprs_upper_lr = np.minimum(mean_tpr_lr + std_tpr_lr, 1)
    tprs_lower_lr = np.maximum(mean_tpr_lr - std_tpr_lr, 0)
    plt.fill_between(mean_fpr, tprs_lower_lr, tprs_upper_lr, color="r", alpha=0.2)

    plt.plot([0, 1], [0, 1], color='navy', lw=1, linestyle='--')
```

```
    plt.title(name)
    plt.legend(loc="lower right")
    plt.savefig(save_name, dpi=300, bbox_inches="tight")
    plt.show()

def fi_plot(fi_df, name, save_name):
    fi_plot_mean = (fi_df[0] + fi_df[1] + fi_df[2] + fi_df[3] + fi_df[4]) / 5
    fi_plot_mean_sorted = fi_plot_mean.sort_values(["Value"], ascending=False).head(25)

    fi_value = pd.DataFrame([fi_df[0].index, np.around(np.mean(fi_df,
axis=0).flatten(), 3),np.around(np.std(fi_df, axis=0).flatten(), 3)]).T
    fi_value.columns = ["name", "fi_df_mean", "fi_df_std"]
    fi_value_sorted = fi_value.sort_values(["fi_df_mean"], ascending=False).head(25)
    Value = pd.DataFrame(fi_value_sorted["fi_df_mean"].astype(str) + "±" +
fi_value_sorted["fi_df_std"].astype(str))

    plt.figure(figsize=(5, 10))
    sns.heatmap(fi_plot_mean_sorted, vmin=0, vmax=1, annot=Value, cmap='Blues', fmt =
'')
    plt.title(name)
    plt.savefig(save_name, dpi=300, bbox_inches="tight")
    plt.show()

if __name__ == '__main__':
    ra_sum = pd.read_csv("RA_surgery_summary v3.0.csv")
    ra_df = ra_sum.drop(["CCP_abnormal", "CRP_abnormal", "RF_abnormal",
"WEST_abnormal", "procedure_d", "index_date"], axis=1)
    ra_x = ra_df.drop(["PatientID", "Surgery_YN", "Surgery_Major_YN"], axis=1)

    ra_df_1 = ra_df[~ra_df["Surgery_Major_YN"].isna()].reset_index(drop=True)
    ra_1_x = ra_df_1.drop(["PatientID", "Surgery_YN", "Surgery_Major_YN"], axis=1)
    mean_fpr = np.linspace(0, 1, 150)

    nominal = ["Sex", "Race", "Ethnicity"]
    continuous = ["AgeInYears", "HeightCM", "BMI", "Biologic_freq", "OralSteroid_freq",
"csDMARD_freq"]

    based_auc_YN, tuned_auc_YN, calibrated_auc_YN, tprs_YN, fi_YN = xgbc(ra_x,
ra_df["Surgery_YN"], "model_3.0/based_YN.pkl", "model_3.0/tuned_YN.pkl",
"model_3.0/calibrated_YN.pkl")
    based_auc_YN_lr, tprs_YN_lr = logistic(ra_x, ra_df["Surgery_YN"],
"model_3.0/based_YN_lr.pkl")
    roc_ccurve(tuned_auc_YN, tprs_YN, based_auc_YN_lr, tprs_YN_lr, "ROC Curve - surgery
at least five years after the index date", 'plot_3.0/ROC_YN_mix.png')
    fi_plot(fi_YN, "Feature Importance - surgery at least five years after the index
date", 'plot_3.0/FI_YN.png')

    based_auc_major, tuned_auc_major, calibrated_auc_major, tprs_major, fi_major =
xgbc(ra_1_x, ra_df_1["Surgery_Major_YN"], "model_3.0/based_major.pkl",
"model_3.0/tuned_major.pkl", "model_3.0/calibrated_major.pkl")
    based_auc_major_lr, tprs_major_lr = logistic(ra_1_x, ra_df_1["Surgery_Major_YN"],
"model_3.0/based_major_lr.pkl")
    roc_ccurve(tuned_auc_major, tprs_major, based_auc_major_lr, tprs_major_lr, "ROC
Curve - surgery level (major versus intermediate)", 'plot_3.0/ROC_major_mix.png')
    fi_plot(fi_major, "Feature Importance - surgery level (major versus intermediate)",
'plot_3.0/FI_major.png')
```