Code:

```
import pandas as pd
import sklearn
import matplotlib.pyplot as plt
import xgboost as xgb
import lightgbm as gbm
import numpy as np
from sklearn import model_selection
from sklearn.metrics import roc_curve, auc, roc_auc_score, average_precision_score, precision_recall_curve
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
import seaborn as sb
from  sklearn.ensemble import RandomForestClassifier
from imblearn.combine import SMOTEENN
from sklearn.linear_model import LogisticRegression
from mlxtend.classifier import StackingCVClassifier
import shap
from sklearn.calibration import calibration_curve

plt.rc('font',family='Arial')

def loadDataset(filePath):
    df = pd.read_csv(filepath_or_buffer=filePath)
    return df

def featureSet(data):
    """ Read the features and labels in the file according to the number of features """
    data_num = len(data)
    XList = []
    yList = []
    for row in range(0, data_num):
        tmp_list = []
        for number in range(0, 35):
            tmp_list.append(data.iloc[row][number])
        XList.append(tmp_list)
    for row in range(0, data_num):
        temp_list = []
        temp_list.append(data.iloc[row][35])
        yList.append(temp_list)
        """ Return 35 features and 1 label """
    return XList, yList
```

```
def loadTestData(filePath):
    """ Read the features and labels in the file according to the number of features """
    data = pd.read_csv(filepath_or_buffer=filePath)
    data_num = len(data)
    XList = []
    yList = []
    for row in range(0, data_num):
        tmp_list = []
        for number in range(0, 35):
            tmp_list.append(data.iloc[row][number])
        XList.append(tmp_list)
    for row in range(0, data_num):
        temp_list = []
        temp_list.append(data.iloc[row][35])
        yList.append(temp_list)
        """ Return 35 features and 1 label """
    return XList, yList


def trainandTestxgb(X_train, y_train, X_test, y_test):
    """     XGBoost module    """
    """ Can use scale_pos_weight or other hyperparameters """
    model = xgb.XGBClassifier(objective="binary:logistic", learning_rate=0.01, n_estimators=400,
max_depth=5, min_child_weight=2)
    X_train = np.array(X_train)
    X_test = np.array(X_test)
    model.fit(X_train, y_train)
    """ SHAP for XGBoost """
    explainer = shap.TreeExplainer(model)
    shap_values = explainer.shap_values(X_test)
    """ feature_names : Feature names array in dataset """
    """ Group level analysis chart of SHAP """
    shap.summary_plot(shap_values, X_test, feature_names=['Feature_1', 'Feature_2',
'Feature_3'])
    shap.summary_plot(shap_values, X_test, plot_type="bar", color='#F2AEA4',
feature_names=['Feature_1', 'Feature_2', 'Feature_3'])
    """ Individual level analysis chart of SHAP """
    Y_shap = np.array(y_test)
    for number in range(0, 200):
        """ Label is 0 (negative) """
        if (Y_shap[number] == 0.0):
            shap.force_plot(explainer.expected_value, shap_values[number, :],
```

```
X_test[number, :], feature_names=['Feature_1', 'Feature_2', 'Feature_3'], matplotlib=True, show=False)
    for number in range(0, 200):
        """ Label is 1 (positive) """
        if (Y_shap[number] == 1.0):
            shap.force_plot(explainer.expected_value, shap_values[number, :], X_test[number, :], feature_names=['Feature_1', 'Feature_2', 'Feature_3'], matplotlib=True, show=False)
            """ Save path """
            plt.savefig("Save_path" + str(number) + ".png")
    """ XGBoost testing results on the testing set """
    test_XGB_pre = model.predict(X_test)
    test_XGB_proba = model.predict_proba(X_test)
    y_test_predprob.append(test_XGB_proba[:, 1])
    """ XGB testing metrics """
    print('The XGBoost accuracy of the Test is:', metrics.accuracy_score(y_test, test_XGB_pre))
    print('The XGBoost F1 of the Test is:', metrics.f1_score(y_test, test_XGB_pre))
    confusion_matrix_result = metrics.confusion_matrix(y_test, test_XGB_pre)
    print('The XGBoost Test confusion matrix result:\n', confusion_matrix_result)
    plt.figure(figsize=(8, 6))
    sb.heatmap(confusion_matrix_result, annot=True, cmap='Blues', fmt='g')
    plt.xlabel('Predicted labels')
    plt.ylabel('True labels')
    plt.show()

    """ For the Weighting model add hyperparameter: scale_pos_weight """
    Basemodel = xgb.XGBClassifier(objective="binary:logistic", learning_rate=0.01, n_estimators=400, max_depth=5, min_child_weight=2, scale_pos_weight=1.5)
    X_train = np.array(X_train)
    X_test = np.array(X_test)
    Basemodel.fit(X_train, y_train)
    Base_XGB_pre = Basemodel.predict(X_test)
    Base_XGB_proba = Basemodel.predict_proba(X_test)
    return metrics.f1_score(y_test, Base_XGB_pre), Base_XGB_proba


def trainandTestgbm(X_train, y_train, X_test, y_test):
    """     LightGBM module    """
    """ can use other hyperparameters """
    clf = gbm.LGBMClassifier(learning_rate=0.05, n_estimators=240, num_leaves=5, max_depth=4, min_child_samples=20)
    clf.fit(X_train, y_train)
    test_LGB_pre = clf.predict(X_test)
    test_LGB_proba = clf.predict_proba(X_test)
```

```
    """ LightGBM testing results on the testing set """
    confusion_matrix_result = metrics.confusion_matrix(y_test, test_LGB_pre)
    print('The Lightgbm accuracy of the Test is:', metrics.accuracy_score(y_test, test_LGB_pre))
    print('The Lightgbm F1 of the Test is:', metrics.f1_score(y_test, test_LGB_pre))
    print('The LightGBM Test confusion matrix result:\n', confusion_matrix_result)
    y_test_predprob.append(LGB_Proba[:, 1])
    plt.figure(figsize=(8, 6))
    sb.heatmap(confusion_matrix_result, annot=True, cmap='Blues', fmt='g')
    plt.xlabel('Predicted labels')
    plt.ylabel('True labels')
    plt.show()
    """ Return F1 score and prediction probability as the input of Weighting """
    return metrics.f1_score(y_test, test_LGB_pre), test_LGB_proba

def trainandTestRF(X_train, y_train, X_test,y_test):
    """      Random Forest module    """
    """ can use other hyperparameters """
    clf = RandomForestClassifier(n_estimators=820, min_samples_leaf=3, max_features=3,
min_samples_split=7, max_depth=21)
    clf.fit(X_train, y_train)
    test_RF_pre = clf.predict(X_test)
    test_RF_proba = clf.predict_proba(X_test)
    """ Random Forest testing results on the testing set """
    ROC_RF = test_RF_proba[:, 1]
    y_test_predprob.append(ROC_RF)
    print('The RF accuracy of the Test is:', metrics.accuracy_score(y_test, test_RF_pre))
    print('The RF F1 of the Test is:', metrics.f1_score(y_test, test_RF_pre))
    confusion_matrix_result = metrics.confusion_matrix(y_test, test_RF_pre)
    print('The RF Test confusion matrix result:\n', confusion_matrix_result)
    plt.figure(figsize=(8, 6))
    sb.heatmap(confusion_matrix_result, annot=True, cmap='Blues', fmt='g')
    plt.xlabel('Predicted labels')
    plt.ylabel('True labels')
    plt.show()
    """ Return F1 score and prediction probability as the input of Weighting """
    return metrics.f1_score(y_test, test_RF_pre), test_RF_proba

def PreDataset(filepath1):
    data1 = np.loadtxt(filepath1, dtype=str, skiprows=1, delimiter=',')
    X, y = data1[:, 1:-1], data1[:, -1]
    """ 70% of training set """
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
    train = np.column_stack((X_train, y_train))
    np.savetxt('File_path_Train', train, fmt='%s', delimiter=',')
```

```
    """ 30% of testing set """
    test = np.column_stack((X_test, y_test))
    np.savetxt('File_path_Test', test, fmt='%s', delimiter=',')
    return X, y

def XGBoost_cv(X_train, y_train):
    """ Use grid search to adjust the hyperparameter in XGBoost """
    X_train = np.array(X_train)
    y_train = np.array(y_train)
    cv_params = {'min_child_weight': [1, 2, 3, 4, 5]}
    """ Some optional hyperparameter """
    other_params = {'scale_pos_weight': 1.5, 'n_estimators': 400, 'learning_rate': 0.01,
'max_depth': 5, 'min_child_weight' : 2}
    model = xgb.XGBClassifier(**other_params)
    """ scoring:F1,accuracy,recall... """
    optimized_GBM = GridSearchCV(estimator=model, param_grid=cv_params, scoring='f1',
cv=5, verbose=1, n_jobs=4)
    optimized_GBM.fit(X_train, y_train)
    evalute_result = optimized_GBM.cv_results_
    print('XGB results of each iteration:{0}'.format(evalute_result))
    print('XGB optimum value of parameter：{0}'.format(optimized_GBM.best_params_))
    print('XGB best model score:{0}'.format(optimized_GBM.best_score_))

def LightGBM_cv(X_train, y_train):
    """ Use grid search to adjust the hyperparameter in LightGBM """
    X_train = np.array(X_train)
    y_train = np.array(y_train)
    cv_params = {'min_child_samples': [18, 20, 22, 24, 26]}
    """ Some optional hyperparameter """
    other_params = {'n_estimators': 240, 'learning_rate': 0.05, 'num_leaves' : 5, 'max_depth': 4,
'min_child_samples' : 20}
    model = gbm.LGBMClassifier(**other_params)
    """ scoring:F1,accuracy,recall... """
    optimized_GBM = GridSearchCV(estimator=model, param_grid=cv_params, scoring='f1',
cv=5, verbose=1, n_jobs=4)
    optimized_GBM.fit(X_train, y_train)
    evalute_result = optimized_GBM.cv_results_
    print('LGB results of each iteration:{0}'.format(evalute_result))
    print('LGB optimum value of parameter：{0}'.format(optimized_GBM.best_params_))
    print('LGB best model score:{0}'.format(optimized_GBM.best_score_))

def RF_cv(X_train, y_train):
    """ Use grid search to adjust the hyperparameter in Random Forest """
    X_train = np.array(X_train)
```

```
    y_train = np.array(y_train)
    cv_params = {'n_estimators': [150, 200, 250, 300, 350]}
    """ Some optional hyperparameter """
    other_params = {"""'n_estimators': 400, 'max_depth': 20, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': 'auto',
                                'n_jobs': 4, 'random_state': 10, 'min_weight_fraction_leaf': 0"""}
    model = RandomForestClassifier(**other_params)
    """ scoring:F1,accuracy,recall... """
    optimized_GBM = GridSearchCV(estimator=model, param_grid=cv_params, scoring='f1', cv=5, verbose=1, n_jobs=4)
    optimized_GBM.fit(X_train, y_train)
    evalute_result = optimized_GBM.cv_results_
    print('RF results of each iteration:{0}'.format(evalute_result))
    print('RF optimum value of parameter：{0}'.format(optimized_GBM.best_params_))
    print('RF best model score:{0}'.format(optimized_GBM.best_score_))

def Weighting_model(XGBoost_Proba, XGB_F1, LGB_Proba, LGB_F1, RF_Proba, RF_F1, y_test):
    """ Weighting Model """
    data_num = len(XGBoost_Proba)
    weight_xgb = XGB_F1/(XGB_F1 + LGB_F1 + RF_F1)
    weight_lgb = LGB_F1/(XGB_F1 + LGB_F1 + RF_F1)
    weight_rf = RF_F1/(XGB_F1 + LGB_F1 + RF_F1)
    """ Weighting the prediction probability of multiple models by F1 score """
    Predict_Weighting = []
    for row in range(0, data_num):
        tmp_list = []
        tmp_list.append(XGBoost_Proba[row][0] * weight_xgb + LGB_Proba[row][0] * weight_lgb + RF_Proba[row][0] * weight_rf)
        tmp_list.append(XGBoost_Proba[row][1] * weight_xgb + LGB_Proba[row][1] * weight_lgb + RF_Proba[row][1] * weight_rf)
        Predict_Weighting.append(tmp_list)
    """ Use the testing result as input to the ROC-PR plot function """
    Predict_Weighting = np.array(Predict_Weighting)
    y_test_predprob.append(Predict_Weighting[:, 1])
    """ Convert the comprehensive prediction probability to the prediction label by 0.5 threshold """
    Predict_Weighting = np.int64(Predict_Weighting[:, 1] > 0.5)
    print('The Weighting accuracy of the Test is:', metrics.accuracy_score(y_test, Predict_Weighting))
    print('The Weighting F1 of the Test is:', metrics.f1_score(y_test, Predict_Weighting))
    confusion_matrix_result = metrics.confusion_matrix(y_test, Predict_Weighting)
    print('The Weighting Test confusion matrix result:\n', confusion_matrix_result)
    print(metrics.classification_report(y_test, Predict_Weighting))
    plt.figure(figsize=(8, 6))
```

```
    sb.heatmap(confusion_matrix_result, annot=True, cmap='Blues',fmt='g')
    plt.xlabel('Predicted labels')
    plt.ylabel('True labels')
    plt.show()

def Draw_ROC_PR(names, y_test_predprob, colors, y_test, dpin=100):
    """
    ROC and PR curves of multiple machine learning models are output to one graph
    Args:
        names: list, Names of multiple models
        y_test_predprob: list, Probability predictors of multiple models
    """
    plt.figure(figsize=(20, 20), dpi=dpin)
    """ ROC """
    for (name, y_predprob, colorname) in zip(names, y_test_predprob, colors):
        fpr, tpr, thresholds = roc_curve(y_test, y_predprob, pos_label=1)
        plt.plot(fpr, tpr, lw=3, label='{} (AUC={:.3f})'.format(name, auc(fpr, tpr)),
color=colorname)
        plt.plot([0, 1], [0, 1], '--', lw=5, color='grey')
        plt.axis('square')
        plt.xlim([0, 1])
        plt.ylim([0, 1])
        plt.xlabel('False Positive Rate', fontsize=20)
        plt.ylabel('True Positive Rate', fontsize=20)
        plt.title('ROC Curve', fontsize=25)
        plt.legend(loc='lower right', fontsize=10)
    plt.grid()
    plt.show()
    """ PR """
    for (name, y_predprob, colorname) in zip(names, y_test_predprob, colors):
        precision, recall, thresholds = precision_recall_curve(y_test, y_predprob,pos_label=1)
        plt.plot(precision, recall, lw=3, label='{} (AUC={:.3f})'.format(name,
average_precision_score(y_test, y_predprob)), color=colorname)
        plt.plot([0, 1], [1, 0], '--', lw=5, color='grey')
        plt.axis('square')
        plt.xlim([0, 1])
        plt.ylim([0, 1])
        plt.xlabel('Recall', fontsize=20)
        plt.ylabel('Precision', fontsize=20)
        plt.title('Precision-Recall Curve', fontsize=25)
        plt.legend(loc='lower right', fontsize=10)
    plt.grid()
    plt.show()
```

```
def change_list_order(list):
    """ Adjust the model order in the list to be consistent with the names """
    list_stack = list[4]
    list_blend = list[5]
    list_weight = list[6]
    list[4] = list_weight
    list[5] = list_stack
    list[6] = list_blend
    return list

def multi_models_calibration_curve(names, y_test_predprob, colors, y_test, dpin=100):
    """Drawing of calibration curves"""
    plt.figure(figsize=(20, 20), dpi=dpin)
    for (name, y_predprob, colorname) in zip(names, y_test_predprob, colors):
        y_means, proba_means = calibration_curve(y_test, y_predprob, strategy='quantile')
        plt.plot(proba_means, y_means, lw=3, label='{}'.format(name), color=colorname)
        plt.plot([0, 1], [0, 1], '--', lw=5, color='grey')
        plt.axis('square')
        plt.xlim([0, 1])
        plt.ylim([0, 1])
        plt.xlabel('Predicted probability', fontsize=20)
        plt.ylabel('True probability', fontsize=20)
        plt.title('Calibration Curve', fontsize=25)
        plt.legend(loc='lower right', fontsize=10)
    plt.grid()
    plt.show()

if __name__ == '__main__':
    """ List of stored prediction probabilities of each model in the testing set """
    y_test_predprob = []
    X, y = PreDataset('Dataset_filepath.csv')
    """ Initialize the SHAP module """
    shap.initjs()
    """ Partition dataset """
    np.set_printoptions(suppress=True)
    """ File reading path of training set and testing set """
    trainFilePath = 'Train_file_path.csv'
    testFilePath = 'Test_file_path.csv'
    data = loadDataset(trainFilePath)
    """ Original Training Set """
    X_train_Raw, y_train_Raw = featureSet(data)
    X_test, y_test = loadTestData(testFilePath)
    """ SMOTENN algorithm enhances and balances training set """
    sm = SMOTEENN()
```

```
    X_train, y_train = sm.fit_resample(X_train_Raw, y_train_Raw)
    X_train = np.array(X_train)
    X_test = np.array(X_test)
    """ Cross validation tuning hyperparameter """
    XGBoost_cv(X_train_Raw, y_train_Raw)
    LightGBM_cv(X_train_Raw, y_train_Raw)
    RF_cv(X_train_Raw, y_train_Raw)
    """ Get the training and testing performance of the benchmark model """
    test_RF_F1, test_RF_proba = trainandTestRF(X_train_Raw, y_train_Raw, X_test, y_test)
    test_XGB_F1, test_XGB_proba = trainandTestxgb(X_train_Raw, y_train_Raw, X_test, y_test)
    test_LGB_F1, test_LGB_proba = trainandTestgbm(X_train_Raw, y_train_Raw, X_test, y_test)
    """ Logistic regression """
    LR = LogisticRegression()
    LR.fit(X_train_Raw, y_train_Raw)
    """ testing """
    test_LR_pre = LR.predict(X_test)
    test_LR_proba = LR.predict_proba(X_test)
    y_test_predprob.append(test_LR_proba[:, 1])
    confusion_matrix_result = metrics.confusion_matrix(y_test, test_LR_pre)
    print('The LR Test confusion matrix result:\n', confusion_matrix_result)
    plt.figure(figsize=(8, 6))
    sb.heatmap(confusion_matrix_result, annot=True, cmap='Blues', fmt='g')
    plt.xlabel('Predicted labels')
    plt.ylabel('True labels')
    plt.show()

    """ Stacking Model """
    """ Use as few hyperparameters as possible to improve generalization performance """
    lr = LogisticRegression()
    RF = RandomForestClassifier(n_estimators=250)
    LGB = gbm.LGBMClassifier(n_estimators=250)
    XGB = xgb.XGBClassifier(n_estimators=200)
    """ 4 base learners and 1 meta learner """
    sclf = StackingCVClassifier(classifiers=[RF, LGB, XGB, lr], meta_classifier=lr, cv=5,
use_probas=True)
    sclf.fit(X_train, y_train)
    """ testing  """
    test_Stack_pre = sclf.predict(X_test)
    test_Stack_proba = sclf.predict_proba(X_test)
    y_test_predprob.append(test_Stack_proba[:, 1])
    print('The Stack accuracy of the Test is:', metrics.accuracy_score(y_test, test_Stack_pre))
    print('The Stack F1 of the Test is:', metrics.f1_score(y_test, test_Stack_pre))
    confusion_matrix_result = metrics.confusion_matrix(y_test, test_Stack_pre)
    print('The Stack Test confusion matrix result:\n', confusion_matrix_result)
```

```
    plt.figure(figsize=(8, 6))
    sb.heatmap(confusion_matrix_result, annot=True, cmap='Blues', fmt='g')
    plt.xlabel('Predicted labels')
    plt.ylabel('True labels')
    plt.show()

    """ Cut-off values """
    color = ['c', 'b', 'g', 'r', 'm', 'y', 'k', 'w']
    """ The position of LAD in the table is the 20th column """
    X_raw = X_test[:, 20]
    Y_raw = test_Stack_proba[:, 1]
    X_raw, Y_raw = zip(*sorted(zip(X_raw, Y_raw)))
    """ Use the fifth degree function to fit the curve, which cannot be lower than the second degree function """
    curve_parameter = np.polyfit(X_raw, Y_raw, 5)
    curve_p = np.poly1d(curve_parameter)
    print(curve_p)
    dd_curve_p = curve_p.deriv().deriv()
    print(dd_curve_p)
    plt.xlabel('LAD')
    plt.ylabel('Probability of AF')
    plt.xlim((30, 70))
    plt.ylim((0, 1))
    plt.plot(X_raw, curve_p(X_raw), c=color[6])
    plt.show()
    plt.xlabel('LAD')
    plt.ylabel('The 2-order derivative')
    plt.xlim((30, 70))
    plt.plot(X_raw, dd_curve_p(X_raw), c=color[6])
    plt.show()
    """ SHAP for Stack """
    explainer = shap.KernelExplainer(sclf.predict_proba, X_test)
    shap_values = explainer.shap_values(X_test)
    """ feature_names : Feature names array in dataset """
    """ Group level analysis chart of SHAP """
    shap.summary_plot(shap_values[1], X_test, feature_names=['Feature_1', 'Feature_2', 'Feature_3'])
    shap.summary_plot(shap_values[1], X_test, plot_type="bar", color='#F2AEA4', feature_names=['Feature_1', 'Feature_2', 'Feature_3'])
    """ Individual level analysis chart of SHAP """
    Y_shap = np.array(y_test)
    for number in range(0, 200):
        if (Y_shap[number] == 1.0):
            shap.force_plot(explainer.expected_value[1], shap_values[1][number, :],
```

```
X_test[number, :], feature_names=['Feature_1', 'Feature_2', 'Feature_3'], matplotlib=True, show=False)
    for number in range(0, 200):
        if (Y_shap[number] == 0.0):
            shap.force_plot(explainer.expected_value[1], shap_values[1][number, :], X_test[number, :], feature_names=['Feature_1', 'Feature_2', 'Feature_3'], matplotlib=True, show=False)
            """ Save path """
            plt.savefig("Save_path" + str(number) + ".png")

    """ Blending Model """
    """ Use as few hyperparameters as possible to improve generalization performance """
    RF = RandomForestClassifier(n_estimators=250)
    LGB = gbm.LGBMClassifier(n_estimators=250)
    XGB = xgb.XGBClassifier(n_estimators=200)
    lr = LogisticRegression()
    """ 4 base learners and 1 meta learner """
    bclf = [lr, RF, LGB, XGB]
    X_t1, X_t2, y_t1, y_t2 = train_test_split(X_train, y_train, test_size=0.7)
    dataset_d1 = np.zeros((X_t2.shape[0], len(bclf)))
    dataset_d2 = np.zeros((X_test.shape[0], len(bclf)))
    for j, clf in enumerate(bclf):
        clf.fit(X_t1, y_t1)
        y_submission = clf.predict_proba(X_t2)[:, 1]
        dataset_d1[:, j] = y_submission
        dataset_d2[:, j] = clf.predict_proba(X_test)[:, 1]
    blend_clf = lr
    blend_clf.fit(dataset_d1, y_t2)
    test_Blend_proba = blend_clf.predict_proba(dataset_d2)
    test_Blend_pre = blend_clf.predict(dataset_d2)
    y_test_predprob.append(test_Blend_proba[:, 1])
    print('The Blend accuracy of the Test is:', metrics.accuracy_score(y_test, test_Blend_pre))
    print('The Blend F1 of the Test is:', metrics.f1_score(y_test, test_Blend_pre))
    confusion_matrix_result = metrics.confusion_matrix(y_test, test_Blend_pre)
    print('The Blend Test confusion matrix result:\n', confusion_matrix_result)
    plt.figure(figsize=(8, 6))
    sb.heatmap(confusion_matrix_result, annot=True, cmap='Blues', fmt='g')
    plt.xlabel('Predicted labels')
    plt.ylabel('True labels')
    plt.show()
    """ SHAP for Blend """
    explainer = shap.KernelExplainer(clf.predict_proba, X_t2)
    shap_values = explainer.shap_values(X_test)
    """ feature_names : Feature names array in dataset """
```

```
    """ Group level analysis chart of SHAP """
    shap.summary_plot(shap_values[1], X_test, feature_names=['Feature_1', 'Feature_2', 'Feature_3'])
    shap.summary_plot(shap_values[1], X_test, plot_type="bar", color='#F2AEA4', feature_names=['Feature_1', 'Feature_2', 'Feature_3'])
    """ Individual level analysis chart of SHAP """
    Y_shap = np.array(y_test)
    for number in range(0, 200):
        if (Y_shap[number] ==  1.0):
            shap.force_plot(explainer.expected_value[1], shap_values[1][number, :], X_test[number, :], feature_names=['Feature_1', 'Feature_2', 'Feature_3'], matplotlib=True, show=False)
    for number in range(0, 200):
        if (Y_shap[number] == 0.0):
            shap.force_plot(explainer.expected_value[1], shap_values[1][number, :], X_test[number, :], feature_names=['Feature_1', 'Feature_2', 'Feature_3'], matplotlib=True, show=False)
            """ Save path """
            plt.savefig("Save_path" + str(number) + ".png")
    """ Weighting Model"""
    Weighting_model(test_XGB_proba, test_XGB_F1, test_LGB_proba, test_LGB_F1, test_RF_proba, test_RF_F1, y_test)
    """ Draw ROC and PR curves of all models in testing """
    names = ['RF',
             'XGBoost',
             'LightGBM',
             'LR',
             'Weight',
             'Stack',
             'Blend',
             ]
    colors = ['crimson',
              'orange',
              'gold',
              'black',
              'mediumseagreen',
              'gray',
              'steelblue',
              ]
    y_test_predprob = change_list_order(y_test_predprob)
    Draw_ROC_PR(names, y_test_predprob, colors, X_test, y_test)
    """ Draw Calibration curves of all models in testing """
    multi_models_calibration_curve(names, y_test_predprob, colors, y_test)
```