**Core code**

Five Fold cross-validation part

# core model

models = {

"xgboost": XGBClassifier(use\_label\_encoder=False,

eval\_metric="logloss", subsample=0.8,

colsample\_bytree=0.8, reg\_lambda=1,

min\_child\_weight=3, max\_depth=7),

"lightGBM": LGBMClassifier(learning\_rate=0.03, subsample=0.8,

colsample\_bytree=0.8, reg\_lambda=1,

min\_child\_weight=3, max\_depth=7),

"randomForest": RandomForestClassifier(),

"decisionTree": DecisionTreeClassifier(),

"logistic": LogisticRegression(max\_iter=1000)

}

# 5 fold

k = 5

kf = StratifiedKFold(n\_splits=k, shuffle=True, random\_state=123)

results = []

for model\_name, model in models.items():

accs, aucs, precisions, recalls, f1s = [], [], [], [], []

for train\_idx, test\_idx in kf.split(csgo[x], csgo[y]):

X\_train, X\_test = csgo.iloc[train\_idx][x], csgo.iloc[test\_idx][x]

y\_train, y\_test = csgo.iloc[train\_idx][y], csgo.iloc[test\_idx][y]

model.fit(X\_train, y\_train)

prob = model.predict\_proba(X\_test)[:, 1]

pred = (prob > 0.5).astype(int)

accs.append(accuracy\_score(y\_test, pred))

aucs.append(roc\_auc\_score(y\_test, prob))

precisions.append(precision\_score(y\_test, pred))

recalls.append(recall\_score(y\_test, pred))

f1s.append(f1\_score(y\_test, pred))

results.append({

"Model": model\_name,

"Accuracy\_mean": np.mean(accs), "Accuracy\_max": np.max(accs), "Accuracy\_min": np.min(accs),

"AUC\_mean": np.mean(aucs), "AUC\_max": np.max(aucs), "AUC\_min": np.min(aucs),

"Precision\_mean": np.mean(precisions), "Precision\_max": np.max(precisions), "Precision\_min": np.min(precisions),

"Recall\_mean": np.mean(recalls), "Recall\_max": np.max(recalls), "Recall\_min": np.min(recalls),

"F1\_mean": np.mean(f1s), "F1\_max": np.max(f1s), "F1\_min": np.min(f1s)

})

XGB+SHAP Part

import shap

import matplotlib.pyplot as plt

from xgboost import XGBClassifier

# XGB Part

final\_model = XGBClassifier(use\_label\_encoder=False,

eval\_metric="logloss", subsample=0.8,

colsample\_bytree=0.8, reg\_lambda=1,

min\_child\_weight=3, max\_depth=7)

final\_model.fit(csgo[x], csgo[y])

# creat SHAP explainer（tree-based explainer）

explainer = shap.Explainer(final\_model, csgo[x])

shap\_values = explainer(csgo[x]) # shap.Explanation

# summary plot

shap.summary\_plot(shap\_values, csgo[x], feature\_names=x, max\_display=20)

# Bar plot

shap.summary\_plot(shap\_values, csgo[x], plot\_type="bar", feature\_names=x, max\_display=20)

#Force plot

shap.force\_plot(explainer.expected\_value, shap\_values.values[0], csgo[x].iloc[0], matplotlib=True)

# Waterfall

shap.plots.waterfall(shap\_values[0])

# Decision Plot

# shap.plots.decision(shap\_values[0])

shap\_values = explainer(csgo[x])

shap.decision\_plot(

explainer.expected\_value,

shap\_values.values,

feature\_names=x )

# Heatmap Plot

shap.plots.heatmap(shap\_values[:N]) # N=sample

#dependence\_plot

feature\_names\_to\_plot = ["Metrics Name"]

for feature\_name in feature\_names\_to\_plot:

print(f" dependence\_plot ：{feature\_name}")

shap.dependence\_plot(feature\_name, shap\_values.values, csgo[x], feature\_names=x)