In [1]:
```
import numpy as np
import pandas as pd
import openpyxl
import matplotlib as mp
import matplotlib.pyplot as plt
import sklearn as sl
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.model_selection import RandomizedSearchCV
from sklearn import neighbors
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from xgboost import XGBClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from numpy import mean
from numpy import std
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import RepeatedKFold
import seaborn as sns
from sklearn.metrics import classification_report
from scipy.stats import loguniform
from sklearn.ensemble import GradientBoostingClassifier
np.random.seed(42)
```

In [2]:
```
AA="C:/Users/ganes/onedrive/Desktop/AI/EC-CO2 REG/SAC-Data.xlsx"
df=pd.read_excel(AA)
df.head(5)
df=pd.read_excel(AA)
df.head(5)
df.shape
```

Out[2]: (480, 26)

In [3]:
```
A=df['NCNF']
B=df["NCNT"]
C = df['NG']
D = df['NC']
E = df["Ag"]
F=df['Bi']
G=df["Co"]
H = df['Cu']
I = df['Fe']
J=df['La']
K=df["Mg"]
L = df['MnO2']
M=df['Ni']
N= df['Sn']
O= df['Sb']
P=df["Pd"]
Q=df["Zn"]
R=df["SAC"]
S=df["PT"]
T=df["KE"]
U=df["NE"]
V=df['VO']
List = [A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V]
List1=pd.concat(List, axis=1, sort=True)
List1.head(5)
List1.shape
```

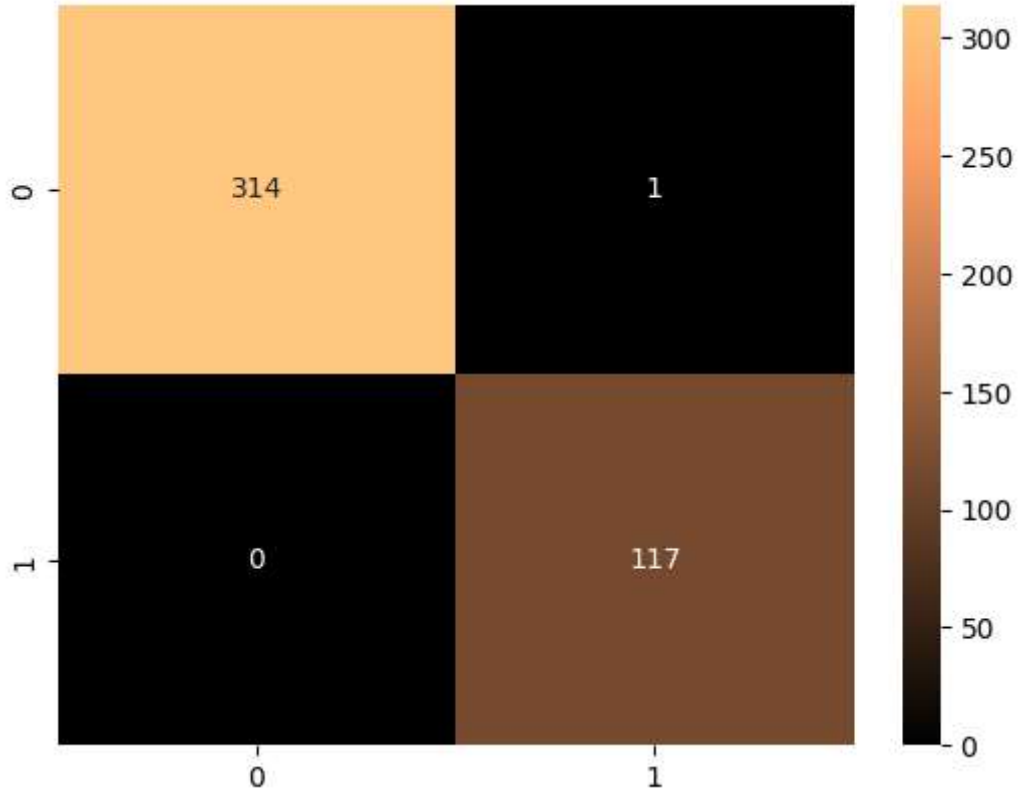
Out[3]: (480, 22)

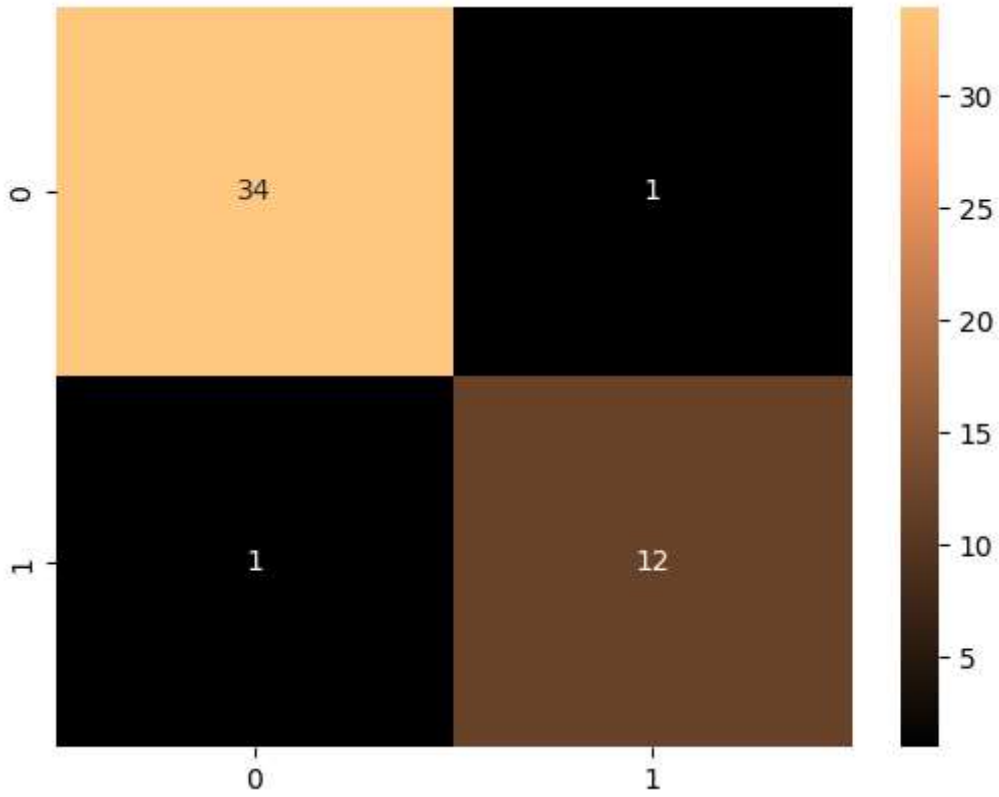
In [4]:
```
X_OS=List1
Y_OS=df["CL"]
```

In [5]:
```
conf_matrix_Train =[]
conf_matrix_Test=[]
mmc_xgbtrain=list()
mmc_xgbtest=list()
score_trainacc, score_testacc, score_trainpre, score_testpre, score_trainrecall, sc
cv = StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
for train_index, test_index in cv.split(X_OS, Y_OS):
    X_Train, X_Test= X_OS.iloc[train_index], X_OS.iloc[test_index]
    Y_Train, Y_Test= Y_OS[train_index], Y_OS[test_index]
    XGB = XGBClassifier(random_state=1, n_estimators=400, learning_rate=0.3,  )
    xgb_model=XGB.fit(X_Train, Y_Train)
    predict_xgbtrain=XGB.predict(X_Train)
    predict_xgbtest=XGB.predict(X_Test)
    Acctrain = metrics.accuracy_score(Y_Train, predict_xgbtrain)
    score_trainacc.append(Acctrain)
    Acctest = metrics.accuracy_score(Y_Test, predict_xgbtest)
    score_testacc.append(Acctest)
    pretrain = metrics.precision_score(Y_Train, predict_xgbtrain, average="macro")
    score_trainpre.append(pretrain)
    pretest = metrics.precision_score(Y_Test, predict_xgbtest, average="macro")
    score_testpre.append(pretest)
    recalltrain = metrics.recall_score(Y_Train, predict_xgbtrain, average="macro")
    score_trainrecall.append(recalltrain)
```

```
    recalltest = metrics.recall_score(Y_Test, predict_xgbtest, average="macro")
    score_testrecall.append(recalltest)
    f1train = metrics.f1_score(Y_Train, predict_xgbtrain, average="macro")
    score_trainf1.append(f1train)
    f1test = metrics.f1_score(Y_Test, predict_xgbtest, average="macro")
    score_testf1.append(f1test)
    conf_matrix1 = confusion_matrix(Y_Train, predict_xgbtrain)
    conf_matrix2 = confusion_matrix(Y_Test, predict_xgbtest)
    mmc_train= metrics.matthews_corrcoef(Y_Train, predict_xgbtrain)
    mmc_xgbtrain.append(mmc_train)
    mmc_test= metrics.matthews_corrcoef(Y_Test, predict_xgbtest)
    mmc_xgbtest.append(mmc_test)


print("train accuracy", mean(score_trainacc))
print("test accuracy", mean(score_testacc))
print("train precision", mean(score_trainpre))
print("test precision", mean(score_testpre))
print("train recall", mean(score_trainrecall))
print("test recall", mean(score_testrecall))
print("train f1", mean(score_trainf1))
print("test f1", mean(score_testf1))
print("train", metrics.classification_report(Y_Train, predict_xgbtrain))
print("test", metrics.classification_report(Y_Test, predict_xgbtest))
print('Matthews correlation coefficient',mean(mmc_xgbtrain))
print('Matthews correlation coefficient',mean(mmc_xgbtest))
sns.heatmap(conf_matrix1, annot=True, cmap='copper', fmt="g")
plt.show()
sns.heatmap(conf_matrix2,annot=True,  cmap='copper', fmt="g")
plt.show()
```

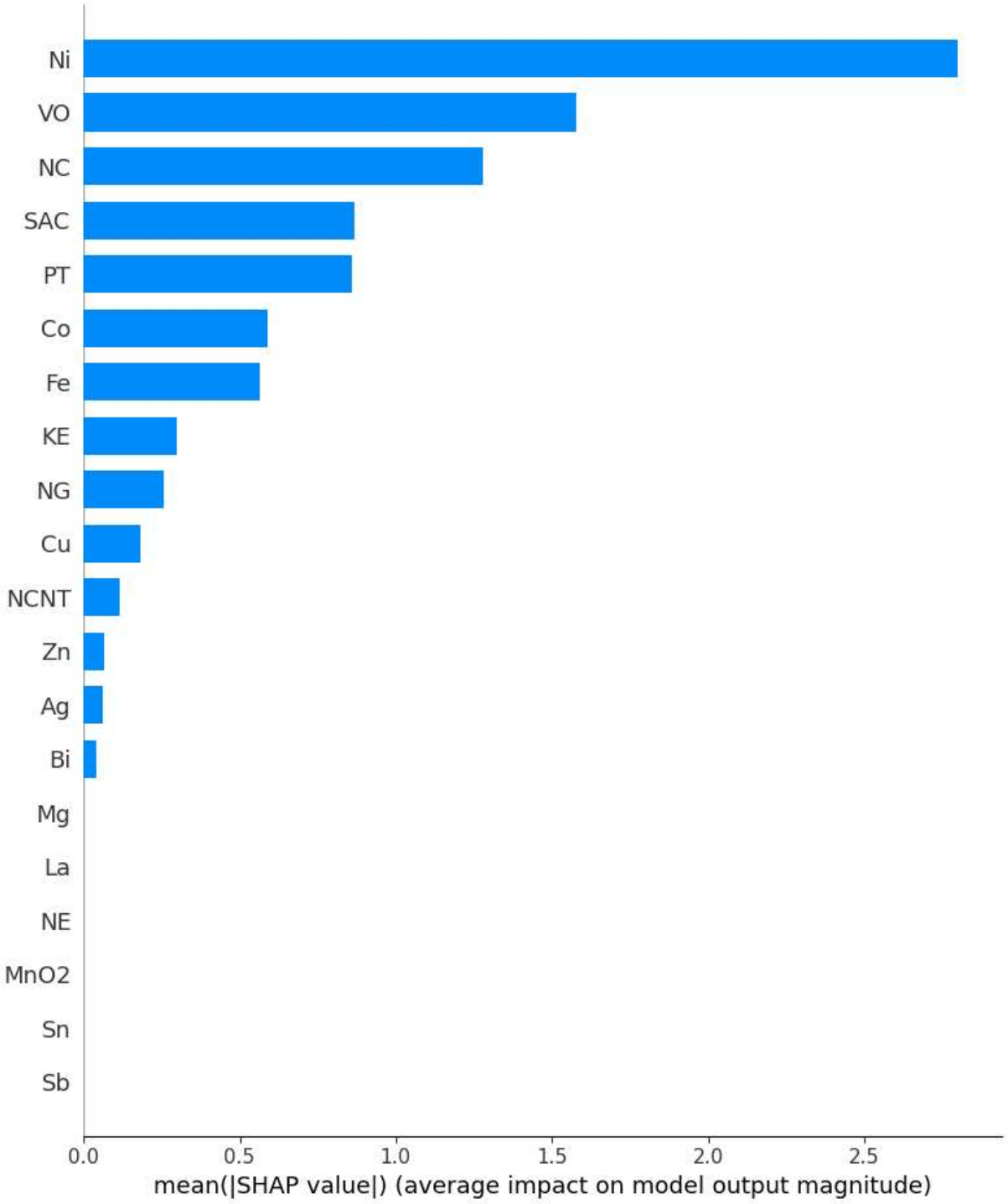
```
train accuracy 0.9981481481481481
test accuracy 0.9291666666666668
train precision 0.9971370474980216
test precision 0.9153140160493102
train recall 0.9981929181929182
test recall 0.9079120879120879
train f1 0.997658801769504
test f1 0.9092307639403205
train               precision    recall  f1-score   support

           0       1.00      1.00      1.00       315
           1       0.99      1.00      1.00       117

    accuracy                           1.00       432
   macro avg       1.00      1.00      1.00       432
weighted avg       1.00      1.00      1.00       432

test               precision    recall  f1-score   support

           0       0.97      0.97      0.97        35
           1       0.92      0.92      0.92        13

    accuracy                           0.96        48
   macro avg       0.95      0.95      0.95        48
weighted avg       0.96      0.96      0.96        48

Matthews correlation coefficient 0.9953278355264713
Matthews correlation coefficient 0.8223075568075149
```



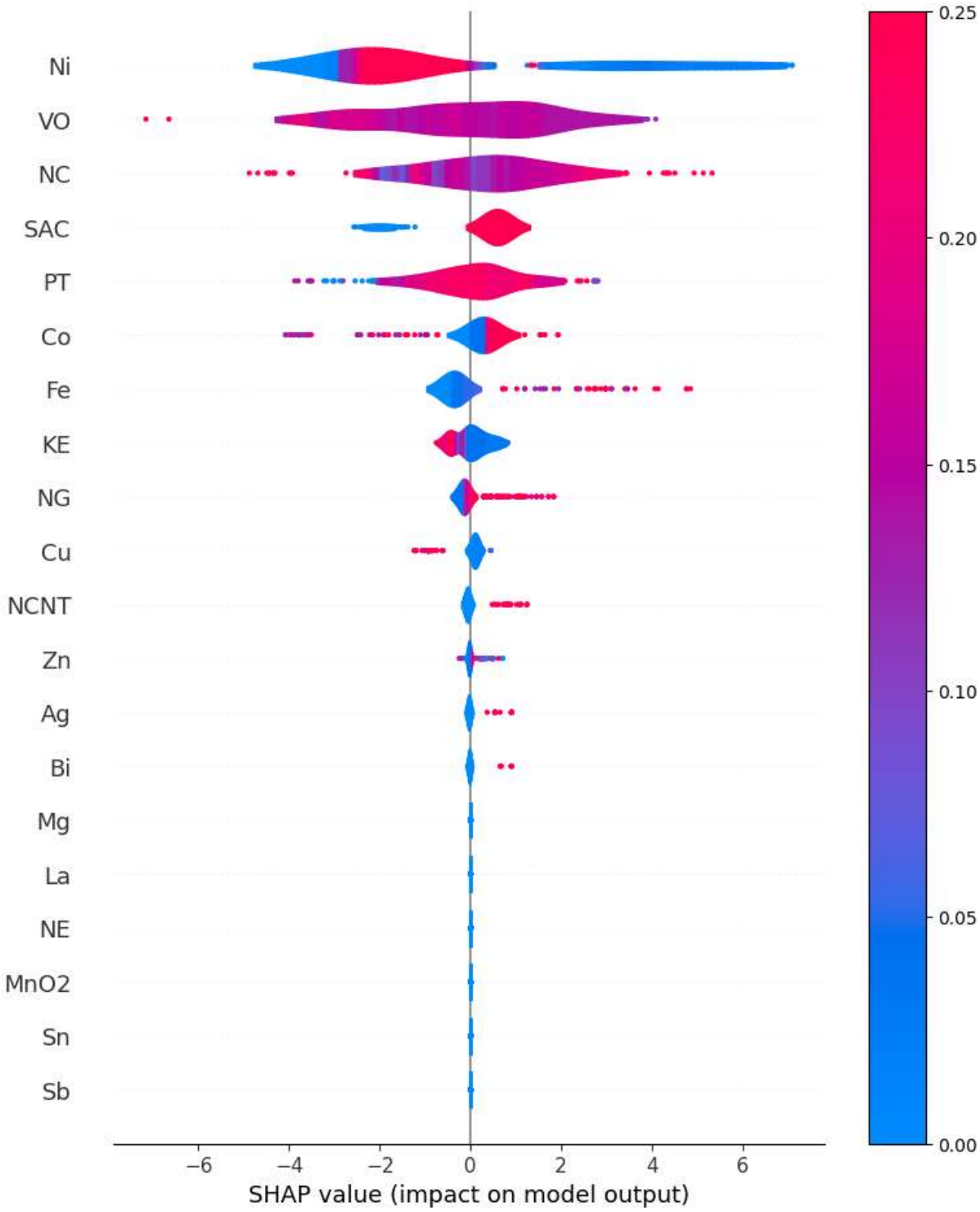


In [6]:
```
import shap
explainer = shap.TreeExplainer(XGB, X_Train)
shap_values_XGB = explainer.shap_values(X_Train)
shap.summary_plot(shap_values_XGB, X_Train, plot_type='bar')
```

```
[22:39:59] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-grou
p-i-0750514818a16474a-1\xgboost\xgboost-ci-windows\src\c_api\c_api.cc:1240: Saving i
nto deprecated binary model format, please consider using `json` or `ubj`. Model for
mat will default to JSON in XGBoost 2.2 if not specified.
```



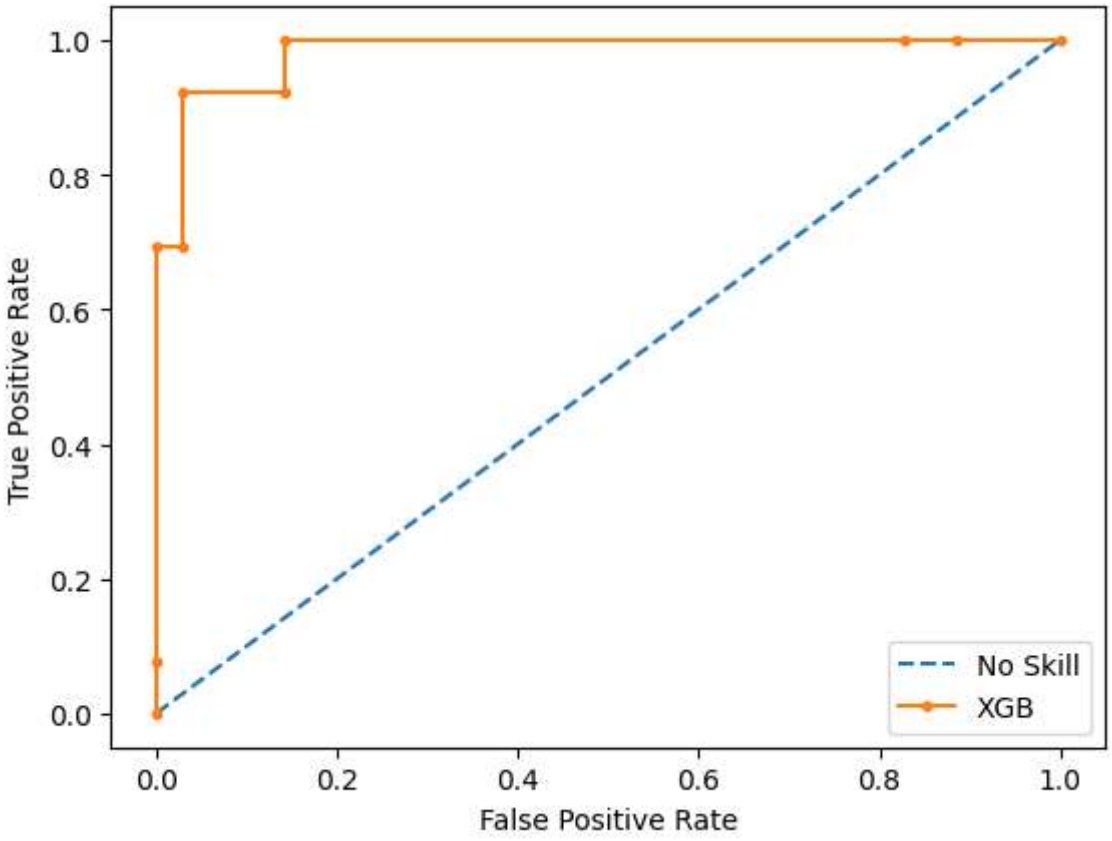
In [7]:
```
shap.summary_plot(shap_values_XGB, X_Train, show = False, color_bar = False, plot_t
plt.colorbar()
plt.show()
```



In [8]:
```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
lr_probs =XGB.predict_proba(X_Test)
lr_probs = lr_probs[:, 1]
ns_probs = [0 for _ in range(len(Y_Test))]
ns_auc = roc_auc_score(Y_Test, ns_probs)
lr_auc = roc_auc_score(Y_Test, lr_probs)
print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('Logistic: ROC AUC=%.3f' % (lr_auc))
ns_fpr, ns_tpr, _ = roc_curve(Y_Test, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(Y_Test, lr_probs)
```

```
# plot the roc curve for the model
plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
plt.plot(lr_fpr, lr_tpr, marker='.', label='XGB')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
# show the plot
plt.show()
```

```
No Skill: ROC AUC=0.500
Logistic: ROC AUC=0.982
```

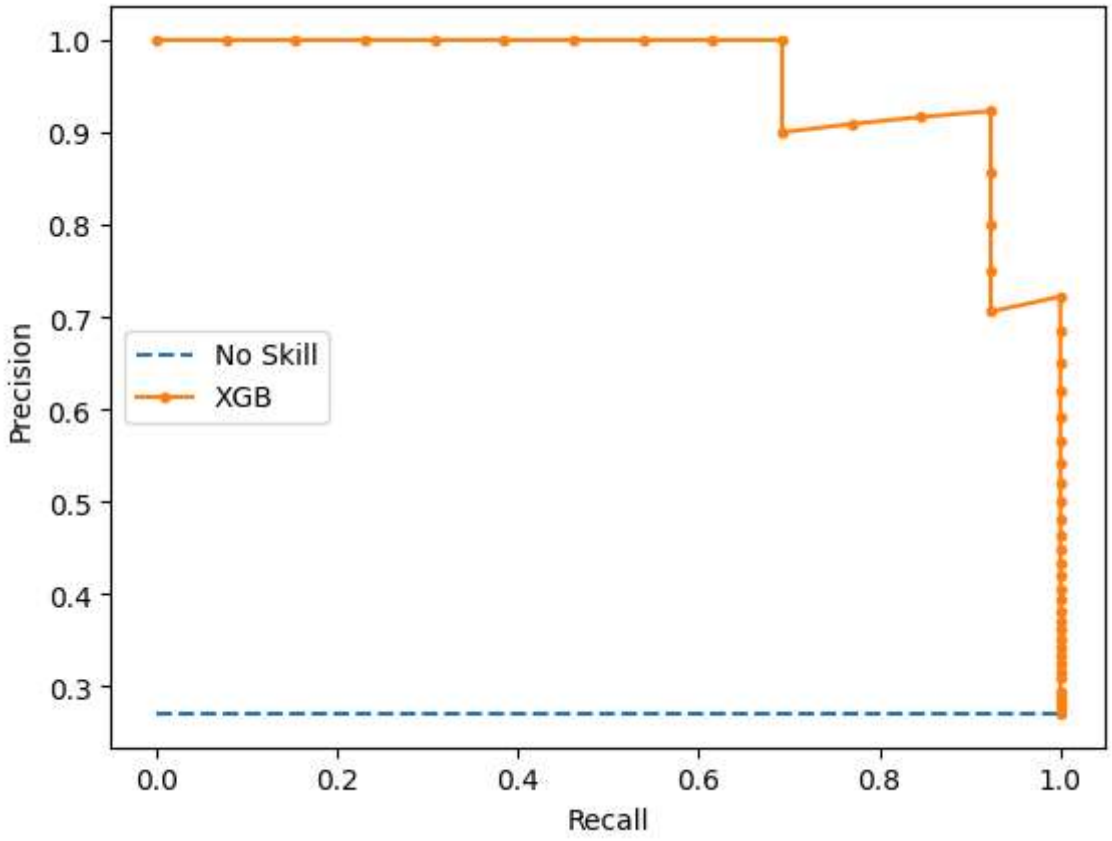


In [9]:

```
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
lr_precision, lr_recall, _ = precision_recall_curve(Y_Test, lr_probs)
yhat=XGB.predict(X_Test)
# calculate scores
lr_f1, lr_auc = f1_score(Y_Test, yhat), auc(lr_recall, lr_precision)
# summarize scores
print('XGB: f1=%.3f auc=%.3f' % (lr_f1, lr_auc))
# plot the precision-recall curves
no_skill = len(Y_Test[Y_Test==1]) / len(Y_Test)
plt.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill')
plt.plot(lr_recall, lr_precision, marker='.', label='XGB')
# axis labels
plt.xlabel('Recall')
plt.ylabel('Precision')
# show the legend
```

```
plt.legend()
# show the plot
plt.show()
```

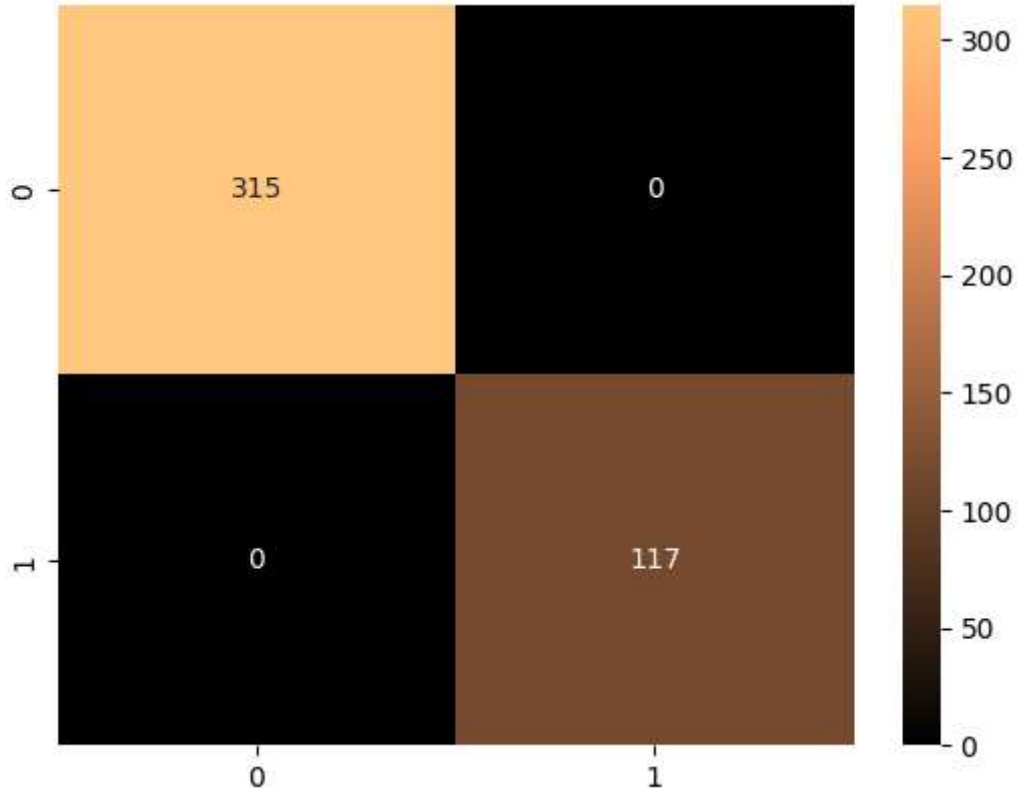
```
XGB: f1=0.923 auc=0.958
```

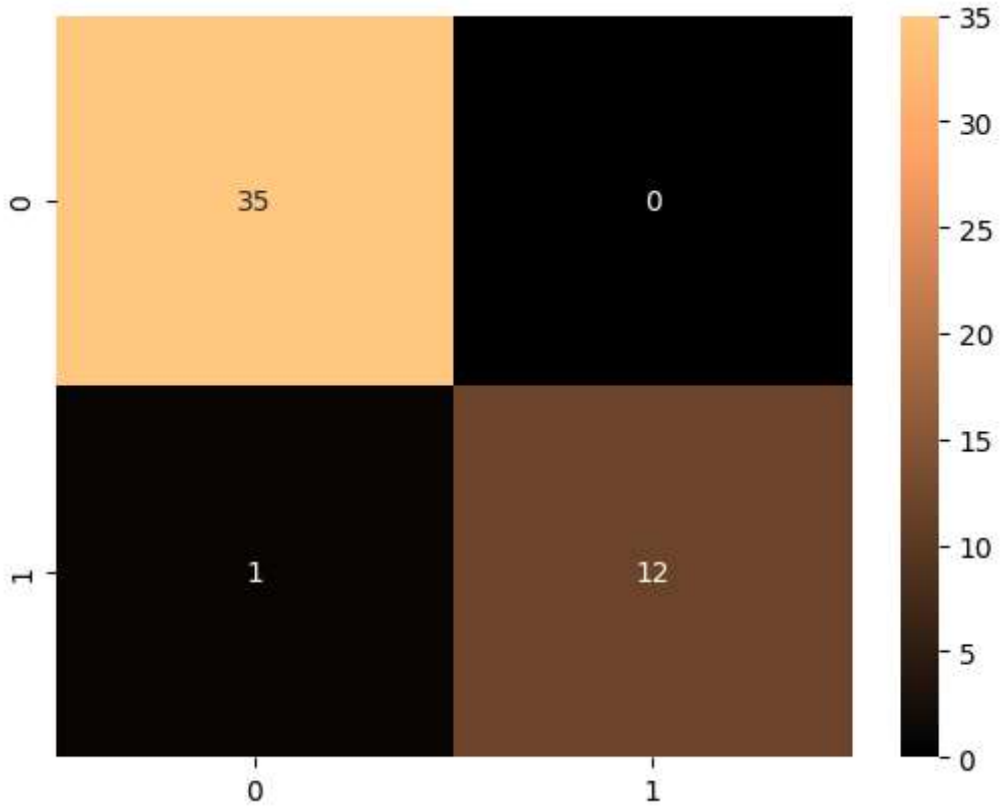


In [10]:
```
conf_matrix_Train =[]
conf_matrix_Test=[]
mmc_dttrain=list()
mmc_dttest=list()
score_trainacc, score_testacc, score_trainpre, score_testpre, score_trainrecall, sc
cv =StratifiedKFold(n_splits=10, random_state=1, shuffle=True)
for train_index, test_index in cv.split(X_OS, Y_OS):
    X_Train, X_Test= X_OS.iloc[train_index], X_OS.iloc[test_index]
    Y_Train, Y_Test= Y_OS[train_index], Y_OS[test_index]
    DT=DecisionTreeClassifier(random_state=42, criterion="entropy" )
    dt_model=DT.fit(X_Train,Y_Train)
    predict_dttrain=DT.predict(X_Train)
    predict_dttest=DT.predict(X_Test)
    Acctrain = metrics.accuracy_score(Y_Train, predict_dttrain)
    score_trainacc.append(Acctrain)
    Acctest = metrics.accuracy_score(Y_Test, predict_dttest)
    score_testacc.append(Acctest)
    pretrain = metrics.precision_score(Y_Train, predict_dttrain, average="macro")
    score_trainpre.append(pretrain)
    pretest = metrics.precision_score(Y_Test, predict_dttest, average="macro")
    score_testpre.append(pretest)
    recalltrain = metrics.recall_score(Y_Train, predict_dttrain, average="macro")
    score_trainrecall.append(recalltrain)
    recalltest = metrics.recall_score(Y_Test, predict_dttest, average="macro")
    score_testrecall.append(recalltest)
```

```
    f1train = metrics.f1_score(Y_Train, predict_dttrain, average="macro")
    score_trainf1.append(f1train)
    f1test = metrics.f1_score(Y_Test, predict_dttest, average="macro")
    score_testf1.append(f1test)
    conf_matrix1 = confusion_matrix(Y_Train, predict_dttrain)
    conf_matrix2 = confusion_matrix(Y_Test, predict_dttest)
    mmc_train= metrics.matthews_corrcoef(Y_Train, predict_dttrain)
    mmc_dttrain.append(mmc_train)
    mmc_test= metrics.matthews_corrcoef(Y_Test, predict_dttest)
    mmc_dttest.append(mmc_test)

print("train accuracy", mean(score_trainacc))
print("test accuracy", mean(score_testacc))
print("train precision", mean(score_trainpre))
print("test precision", mean(score_testpre))
print("train recall", mean(score_trainrecall))
print("test recall", mean(score_testrecall))
print("train f1", mean(score_trainf1))
print("test f1", mean(score_testf1))
print("train", metrics.classification_report(Y_Train, predict_dttrain))
print("test", metrics.classification_report(Y_Test, predict_dttest))
print('Matthews correlation coefficient Train',mean(mmc_dttrain))
print('Matthews correlation coefficient Test',mean(mmc_dttest))
sns.heatmap(conf_matrix1, annot=True, cmap='copper', fmt="g")
plt.show()
sns.heatmap(conf_matrix2,annot=True,  cmap='copper', fmt="g")
plt.show()
```

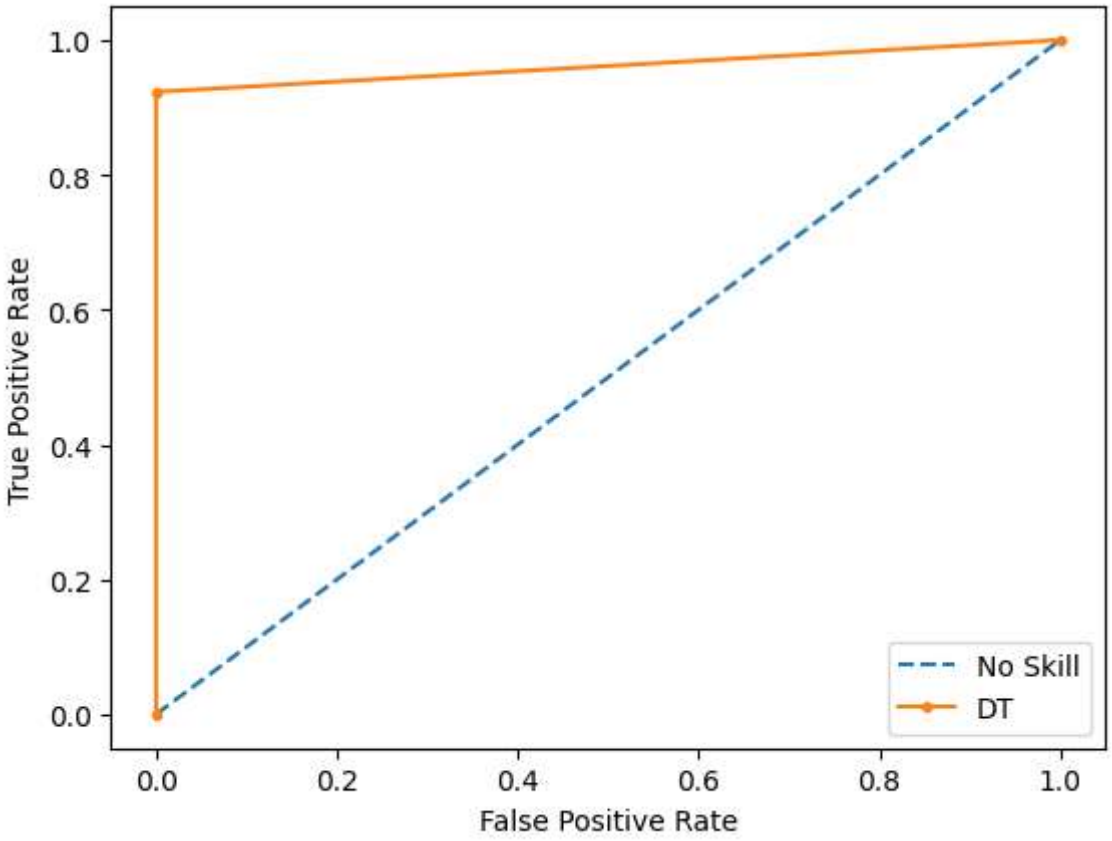
```
train accuracy 1.0
test accuracy 0.9145833333333334
train precision 1.0
test precision 0.9011013507807805
train recall 1.0
test recall 0.8882417582417583
train f1 1.0
test f1 0.8895331418479386
train               precision    recall  f1-score   support

           0       1.00      1.00      1.00       315
           1       1.00      1.00      1.00       117

    accuracy                           1.00       432
   macro avg       1.00      1.00      1.00       432
weighted avg       1.00      1.00      1.00       432

test               precision    recall  f1-score   support

           0       0.97      1.00      0.99        35
           1       1.00      0.92      0.96        13

    accuracy                           0.98        48
   macro avg       0.99      0.96      0.97        48
weighted avg       0.98      0.98      0.98        48

Matthews correlation coefficient Train 1.0
Matthews correlation coefficient Test 0.7871328798688945
```





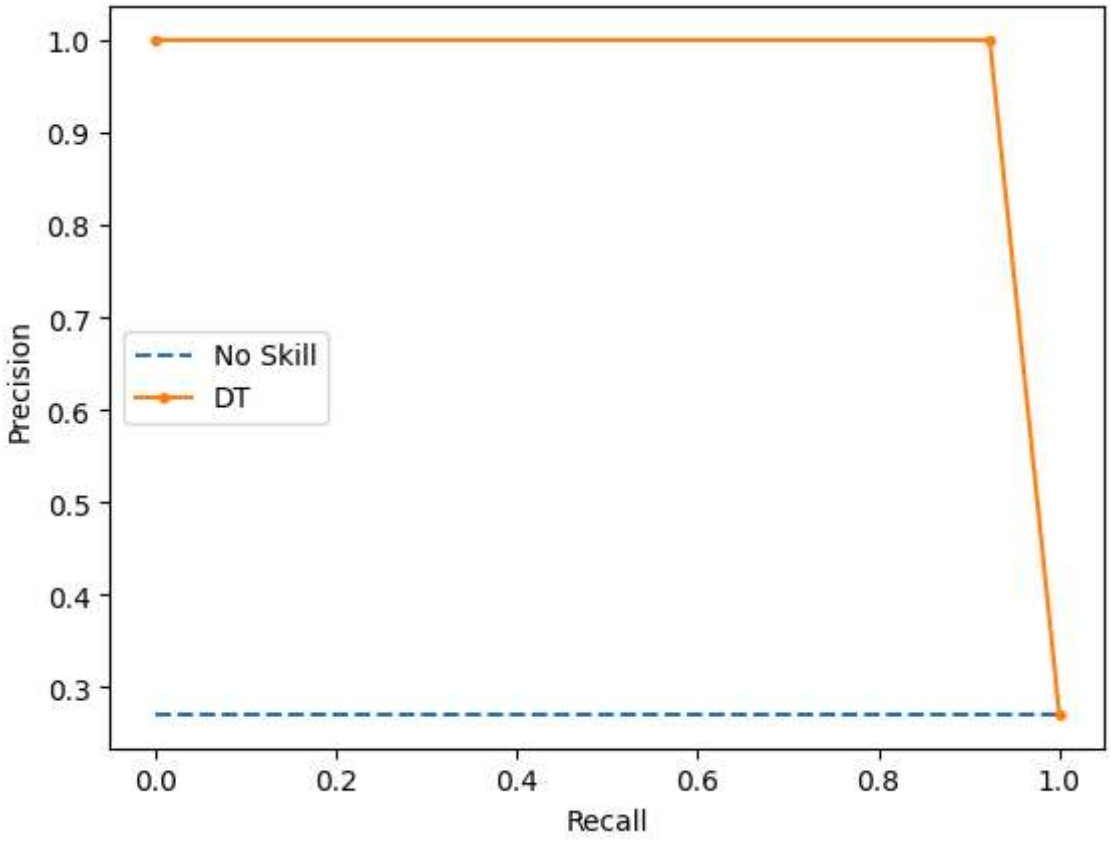
In [11]:
```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
lr_probs =DT.predict_proba(X_Test)
lr_probs = lr_probs[:, 1]
ns_probs = [0 for _ in range(len(Y_Test))]
ns_auc = roc_auc_score(Y_Test, ns_probs)
lr_auc = roc_auc_score(Y_Test, lr_probs)
print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('Logistic: ROC AUC=%.3f' % (lr_auc))
ns_fpr, ns_tpr, _ = roc_curve(Y_Test, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(Y_Test, lr_probs)
# plot the roc curve for the model
plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
plt.plot(lr_fpr, lr_tpr, marker='.', label='DT')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
# show the plot
plt.show()
```

```
No Skill: ROC AUC=0.500
Logistic: ROC AUC=0.962
```



In [15]:
```
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import f1_score
from sklearn.metrics import auc
lr_precision, lr_recall, _ = precision_recall_curve(Y_Test, lr_probs)
yhat=DT.predict(X_Test)
# calculate scores
lr_f1, lr_auc = f1_score(Y_Test, yhat), auc(lr_recall, lr_precision)
# summarize scores
print('DT: f1=%.3f auc=%.3f' % (lr_f1, lr_auc))
# plot the precision-recall curves
no_skill = len(Y_Test[Y_Test==1]) / len(Y_Test)
plt.plot([0, 1], [no_skill, no_skill], linestyle='--', label='No Skill')
plt.plot(lr_recall, lr_precision, marker='.', label='DT')
# axis labels
plt.xlabel('Recall')
plt.ylabel('Precision')
# show the legend
plt.legend()
# show the plot
plt.show()
```

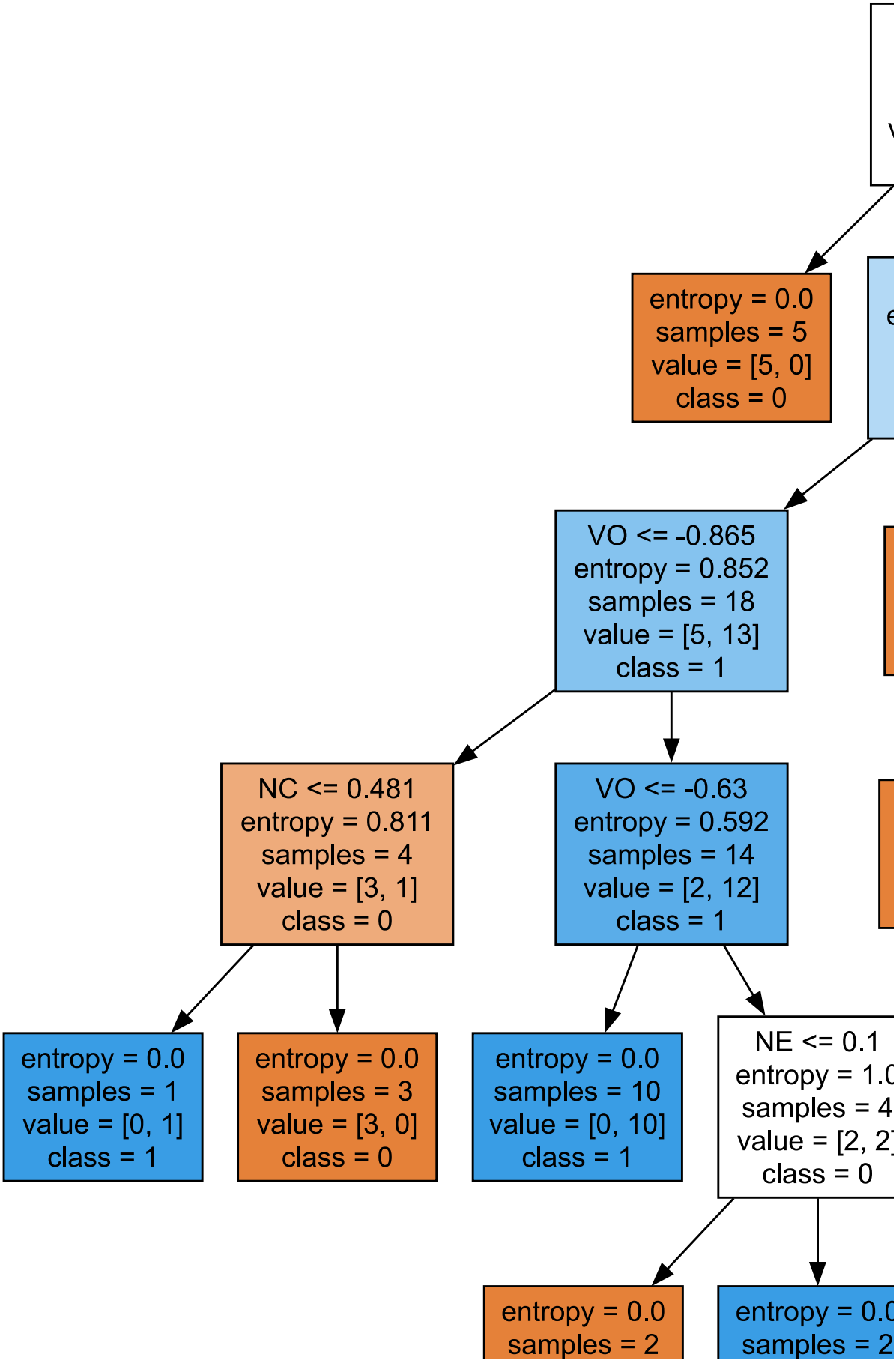
```
DT: f1=0.960 auc=0.972
```



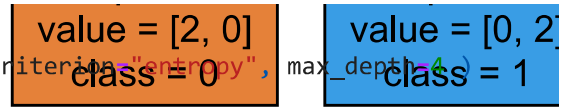
In [16]:
```
from matplotlib import pyplot as plt
from sklearn import tree
from sklearn.tree import export_graphviz
import graphviz
from io import StringIO
```

In [17]:
```
DT=DecisionTreeClassifier(random_state=42, criterion="entropy", )
DT.fit(X_Train,Y_Train)
fn=['NCNF', "NCNT", 'NG', 'NC', "Ag", 'Bi', "Co", 'Cu', 'Fe', 'La', "Mg", 'MnO2', '
cn=["0", "1"]
dot_data = tree.export_graphviz(DT,  feature_names = fn, class_names=cn,  filled =
graph = graphviz.Source(dot_data, format="png")
graph
```

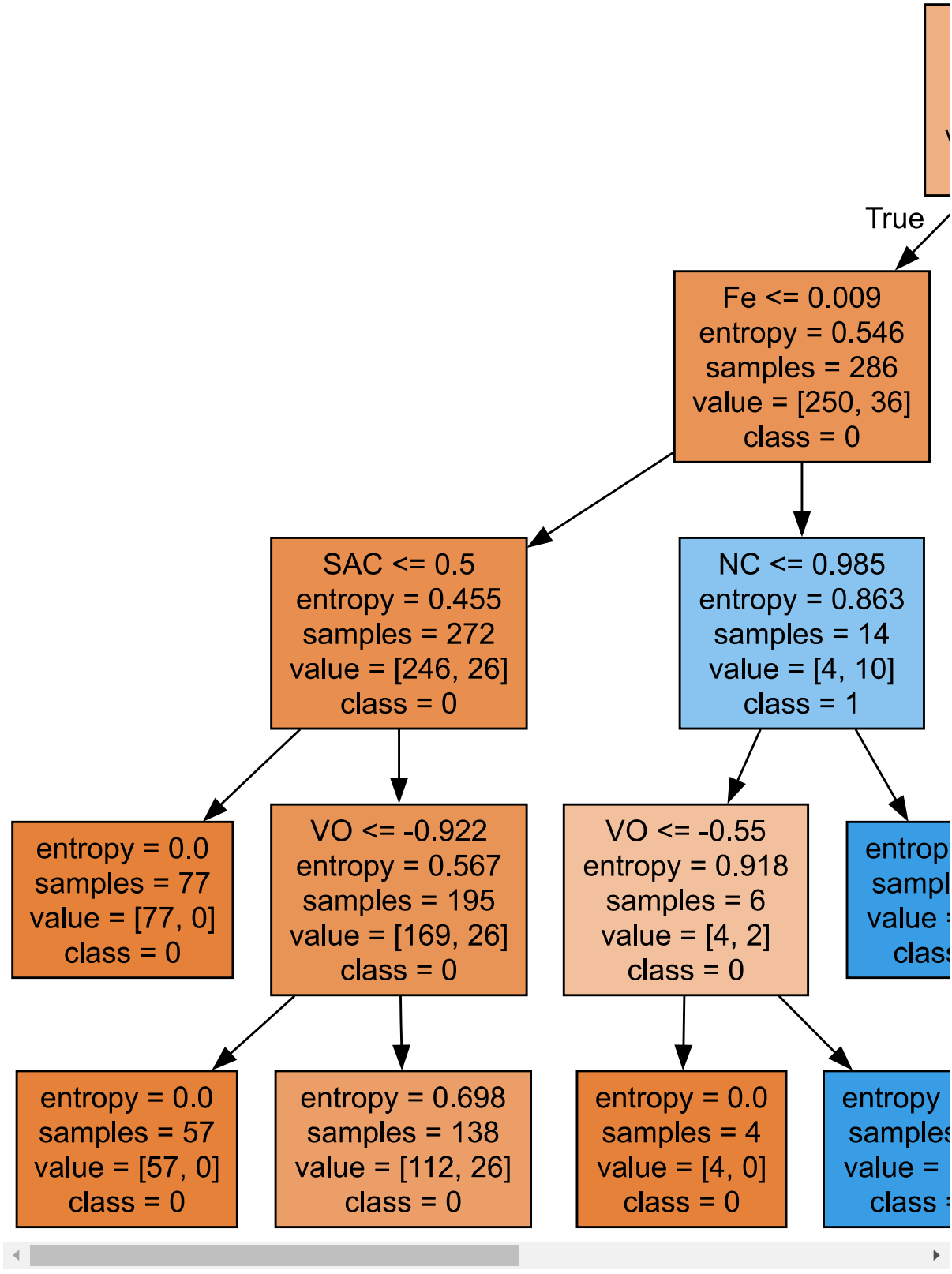
Out[17]:


entropy = 0.0
samples = 5
value = [5, 0]
class = 0
VO <= -0.865
entropy = 0.852
samples = 18
value = [5, 13]
class = 1
NC <= 0.481
entropy = 0.811
samples = 4
value = [3, 1]
class = 0
VO <= -0.63
entropy = 0.592
samples = 14
value = [2, 12]
class = 1
entropy = 0.0
samples = 1
value = [0, 1]
class = 1
entropy = 0.0
samples = 3
value = [3, 0]
class = 0
entropy = 0.0
samples = 10
value = [0, 10]
class = 1
NE <= 0.1
entropy = 0.0
samples = 2
entropy = 0.0
samples = 2

In [18]:
```
DT=DecisionTreeClassifier(random_state=42, criterion="entropy", max_depth=4)
DT.fit(X_Train,Y_Train)
fn=['NCNF', "NCNT", 'NG', 'NC', "Ag", 'Bi', "Co", 'Cu', 'Fe', 'La', "Mg", 'MnO2', '
cn=["0", "1"]
dot_data = tree.export_graphviz(DT,  feature_names = fn, class_names=cn,  filled =
graph = graphviz.Source(dot_data, format="png")
graph
```



Out[18]:


True
Fe <= 0.009
entropy = 0.546
samples = 286
value = [250, 36]
class = 0
SAC <= 0.5
entropy = 0.455
samples = 272
value = [246, 26]
class = 0
NC <= 0.985
entropy = 0.863
samples = 14
value = [4, 10]
class = 1
entropy = 0.0
samples = 77
value = [77, 0]
class = 0
VO <= -0.922
entropy = 0.567
samples = 195
value = [169, 26]
class = 0
VO <= -0.55
entropy = 0.918
samples = 6
value = [4, 2]
class = 0
entropy = 0.0
samples = 57
value = [57, 0]
class = 0
entropy = 0.698
samples = 138
value = [112, 26]
class = 0
entropy = 0.0
samples = 4
value = [4, 0]
class = 0