```
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from xgboost import XGBClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import PowerTransformer
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score, confusion_matrix,classification_report,roc_auc_score, roc_curve

df = pd.read_csv('../creditcard data/creditcard.csv')
classes = df['Class'].value_counts()
print(classes)
normal_share = round((classes[0]/df['Class'].count()*100),2)
print("Normal share =", normal_share)
fraud_share = round((classes[1]/df['Class'].count()*100),2)
print("Fraud share=", fraud_share)
sns.countplot(x='Class', data=df)
plt.title('Number of fraudulent vs non-fraudulent transactions')
#plt.show()

# Creating fraudulent dataframe
data_fraud = df[df['Class'] == 1]
# Creating non fraudulent dataframe
data_non_fraud = df[df['Class'] == 0]
# Distribution plot
plt.figure(figsize=(8,5))
ax = sns.distplot(data_fraud['Amount'],label='fraudulent',hist=False)
ax = sns.distplot(data_non_fraud['Amount'],label='non fraudulent',hist=False)
ax.set(xlabel='Transaction Amount')
#plt.show()

```

```python
# Putting feature variables into X
X = df.drop(['Class'], axis=1)
# Putting target variable to y
y = df['Class']
# Splitting data into train and test set 80:20
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, test_size=0.2, random_state=100)

# Standardization method
# Instantiate the Scaler
scaler = StandardScaler()
# Fit the data into scaler and transform
X_train['Amount'] = scaler.fit_transform(X_train[['Amount']])
print(X_train.head())

# Transform the test set
X_test['Amount'] = scaler.transform(X_test[['Amount']])
print(X_test.head())

# Plotting the distribution of the variables (skewness) of all the columns
cols = X_train.columns
k = 0
plt.figure(figsize=(17, 28))
for col in cols:
 k = k + 1
 plt.subplot(6, 5, k)
 sns.distplot(X_train[col])
 plt.title(col + ' Skewness: ' + str(X_train[col].skew()))
plt.tight_layout()
#plt.show()

# Instantiate the powertransformer
pt = PowerTransformer(method='yeo-johnson', standardize=True,
copy=False)
# Fit and transform the PT on training data
X_train[cols] = pt.fit_transform(X_train)
```

```python
# Transform the test set
X_test[cols] = pt.transform(X_test)
# Plotting the distribution of the variables (
skewness) of all the columns
k=0
plt.figure(figsize=(17,28))
for col in cols :
 k=k+1
 plt.subplot(6, 5,k)
 sns.distplot(X_train[col])
 plt.title(col+' '+str(X_train[col].skew()))
#plt.show()

# Instantiate SMOTE
sm = SMOTE(random_state=27)
# Fitting SMOTE to the train set
X_train_smote, y_train_smote = sm.fit_resample(
X_train, y_train)
print('Before SMOTE oversampling X_train shape=',
X_train.shape)
print('After SMOTE oversampling X_train shape=',
X_train_smote.shape)

# hyperparameter tuning with XGBoost
# creating a KFold object
folds = 3
# specify range of hyperparameters
param_grid = {'learning_rate': [0.2, 0.6], '
subsample': [0.3, 0.6, 0.9]}
# specify model
xgb_model = XGBClassifier(max_depth=2, n_estimators=
200)
# set up GridSearchCV()
model_cv = GridSearchCV(estimator = xgb_model,
param_grid = param_grid, scoring= 'roc_auc', cv =
folds, verbose = 1, return_train_score=True)
# fit the model
model_cv.fit(X_train_smote, y_train_smote)

# cv results
cv_results = pd.DataFrame(model_cv.cv_results_)
```

```python
print(cv_results)

# # plotting
plt.figure(figsize=(16,6))
param_grid = {'learning_rate': [0.2, 0.6], 'subsample': [0.3, 0.6, 0.9]}
for n, subsample in enumerate(param_grid['subsample']):

 # subplot 1/n
 plt.subplot(1,len(param_grid['subsample']), n+1)
 df = cv_results[cv_results['param_subsample']==subsample]
 plt.plot(df["param_learning_rate"], df["mean_test_score"])
 plt.plot(df["param_learning_rate"], df["mean_train_score"])
 plt.xlabel('learning_rate')
 plt.ylabel('AUC')
 plt.title("subsample={0}".format(subsample))
 plt.ylim([0.60, 1])
 plt.legend(['test score', 'train score'], loc='upper left')
 plt.xscale('log')

var = model_cv.best_params_
print(var)

# chosen hyperparameters
# 'objective':'binary:logistic' outputs probability rather than label, which we need for calculating auc
params = {'learning_rate': 0.6, 'max_depth': 2, 'n_estimators': 200, 'subsample': 0.9, 'objective': 'binary:logistic'}
# fit model on training data
xgb_bal_smote_model = XGBClassifier(params=params)
xgb_bal_smote_model.fit(X_train_smote, y_train_smote)
path = '../models'
model_filename = 'xgb_bal_smote_model.bin'
model_filepath = os.path.join(path, model_filename)
```

```python
xgb_bal_smote_model.save_model(model_filepath)

# Predictions on the train set
y_train_pred = xgb_bal_smote_model.predict(
X_train_smote)
# Confusion matrix
confusion = confusion_matrix(y_train_smote,
y_train_pred)
print(confusion)

TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
# Accuracy
print("Accuracy:-", accuracy_score(y_train_smote,
y_train_pred))
# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))
# Specificity
print("Specificity:-", TN / float(TN+FP))

print(classification_report(y_train_smote,
y_train_pred))

# Predicted probability
y_train_pred_proba = xgb_bal_smote_model.
predict_proba(X_train_smote)[:,1]
# roc_auc
auc = roc_auc_score(y_train_smote,
y_train_pred_proba)
print(auc)

# ROC Curve function
def draw_roc( actual, probs ):
 fpr, tpr, thresholds = roc_curve( actual, probs,
drop_intermediate = False )
 auc_score = roc_auc_score( actual, probs )
 plt.figure(figsize=(5, 5))
 plt.plot( fpr, tpr, label='ROC curve (area = %0.2f
)' % auc_score )
```

```
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()
    return None

# Plot the ROC curve
draw_roc(y_train_smote, y_train_pred_proba)

# Predictions on the test set
y_test_pred = xgb_bal_smote_model.predict(X_test)
# Confusion matrix
confusion = confusion_matrix(y_test, y_test_pred)
print(confusion)

TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
# Accuracy
print("Accuracy:-",accuracy_score(y_test, y_test_pred))
# Sensitivity
print("Sensitivity:-",TP / float(TP+FN))
# Specificity
print("Specificity:-", TN / float(TN+FP))

# classification_report
print(classification_report(y_test, y_test_pred))

# Predicted probability
y_test_pred_proba = xgb_bal_smote_model.predict_proba(X_test)[:,1]
# roc_auc
auc = roc_auc_score(y_test, y_test_pred_proba)
```

```python
print(auc)

# Plot the ROC curve
draw_roc(y_test, y_test_pred_proba)
```