```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, precision_recall_curve, auc, confusion_matrix
from sklearn.pipeline import Pipeline

from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as ImbPipeline

import xgboost as xgb
import shap
import joblib

RANDOM_STATE = 42
```

```
df = pd.read_csv("creditcard.csv")   # change path if needed
print("Shape:", df.shape)
print(df.head())
print("Class distribution:\n", df['Class'].value_counts(), "\nNormalized:\n", df['Class'].value_counts(normalize=True))
```

```
Shape: (284807, 31)
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  ...       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  ... -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  ... -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  ...  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  ... -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  ... -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
Class distribution:
 Class
0    284315
1       492
Name: count, dtype: int64
Normalized:
 Class
0    0.998273
1    0.001727
Name: proportion, dtype: float64
```

```
data = df.copy()
if 'Amount' in data.columns:
    data['Amount_scaled'] = (data['Amount'] - data['Amount'].mean()) / (data['Amount'].std() + 1e-9)
    # optional: drop original Amount
    data = data.drop(columns=['Amount'])
if 'Time' in data.columns:
    data['Hour'] = (data['Time'] // 3600) % 24
    # drop Time
    data = data.drop(columns=['Time'])

print("After FE shape:", data.shape)
```

```
After FE shape: (284807, 31)
```

```
X = data.drop(columns=['Class'])
y = data['Class']

# If you have a column 'date' or 'timestamp', use it for a time split. Here we'll do train/test by time if 'date' exists:
if 'date' in data.columns or 'timestamp' in data.columns:
    # Example: sort by timestamp and split (adjust column name to your dataset)
    ts_col = 'date' if 'date' in data.columns else 'timestamp'
    sorted_idx = data.sort_values(ts_col).index
    # 80% train, 20% test by time
    split_point = int(0.8 * len(sorted_idx))
    train_idx = sorted_idx[:split_point]
    test_idx = sorted_idx[split_point:]
    X_train, X_test = X.loc[train_idx], X.loc[test_idx]
    y_train, y_test = y.loc[train_idx], y.loc[test_idx]
else:
    # fallback: stratified random split (keeps class ratios)
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.25, random_state=RANDOM_STATE, stratify=y
    )

print("Train shape:", X_train.shape, "Test shape:", X_test.shape)
print("Train fraud ratio:", y_train.mean(), "Test fraud ratio:", y_test.mean())
```

```
Train shape: (213605, 30) Test shape: (71202, 30)
Train fraud ratio: 0.0017274876524425926 Test fraud ratio: 0.0017274795651807534
```

```
smote = SMOTE(sampling_strategy=0.1, random_state=RANDOM_STATE)
# sampling_strategy=0.1 => minority becomes 10% of majority (tune this)

xgb_clf = xgb.XGBClassifier(
    n_estimators=200,
    max_depth=6,
    learning_rate=0.05,
    use_label_encoder=False,
    eval_metric='logloss',
    random_state=RANDOM_STATE,
    n_jobs=-1
)

pipe = ImbPipeline(steps=[
    ('smote', smote),
    ('scaler', StandardScaler()),   # scales numeric features
    ('clf', xgb_clf)
])

# Quick fit
pipe.fit(X_train, y_train)
```

```
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:199: UserWarning: [14:41:25] WARNING: /workspace/src/learner.cc:79
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

Pipeline

SMOTE

```
SMOTE(random_state=42, sampling_strategy=0.1)
```

StandardScaler

```
StandardScaler()
```

XGBClassifier

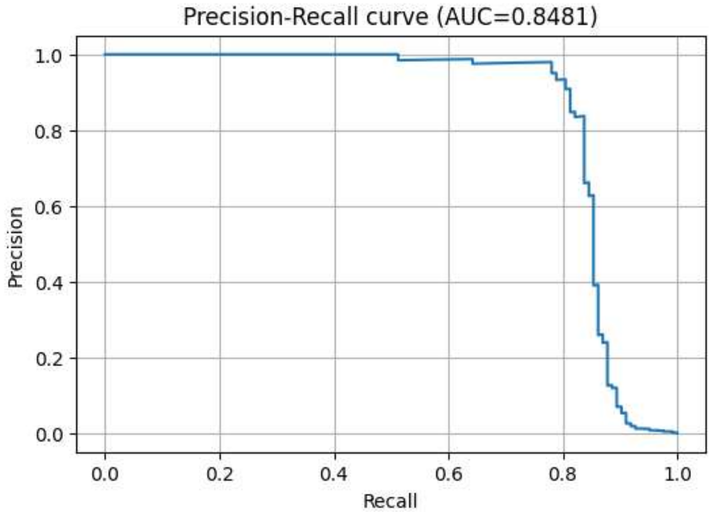
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, feature_weights=None, gamma=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.05, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=6, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=200, n_jobs=-1,
```

```
y_proba = pipe.predict_proba(X_test)[:, 1]   # probability of fraud
# default threshold 0.5 (but we'll tune)
y_pred_default = (y_proba >= 0.5).astype(int)

print("Classification report (threshold=0.5):")
print(classification_report(y_test, y_pred_default, digits=4))

cm = confusion_matrix(y_test, y_pred_default)
print("Confusion matrix:\n", cm)
```

```
Classification report (threshold=0.5):
              precision    recall  f1-score   support

           0     0.9997    0.9993    0.9995     71079
           1     0.6776    0.8374    0.7491       123

    accuracy                         0.9990     71202
   macro avg     0.8387    0.9184    0.8743     71202
weighted avg     0.9992    0.9990    0.9991     71202

Confusion matrix:
 [[71030    49]
 [   20   103]]
```

```
prec, rec, thresh = precision_recall_curve(y_test, y_proba)
pr_auc = auc(rec, prec)
print(f"PR AUC: {pr_auc:.4f}")

plt.figure(figsize=(6,4))
plt.plot(rec, prec)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title(f'Precision-Recall curve (AUC={pr_auc:.4f})')
plt.grid(True)
plt.show()
```

```
PR AUC: 0.8481
```



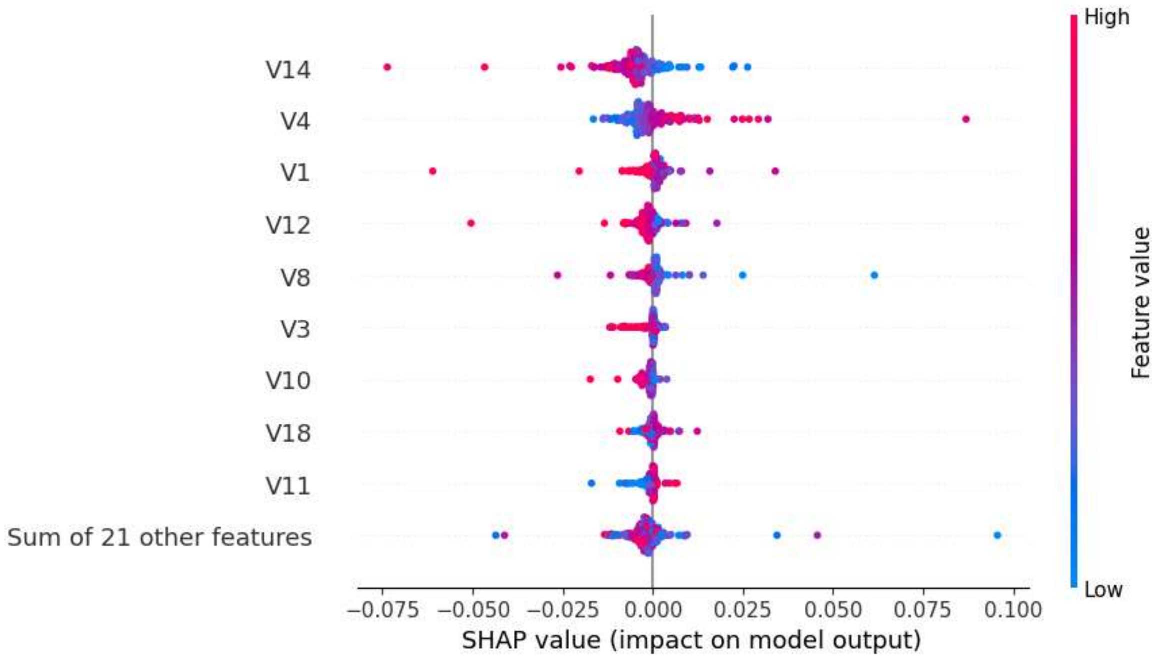

```
target_recall = 0.90
idx = np.argmax(rec >= target_recall)
chosen_thresh = thresh[idx] if idx < len(thresh) else 0.5
print("Chosen threshold for recall>=%.2f -> %.4f" % (target_recall, chosen_thresh))

y_pred_tuned = (y_proba >= chosen_thresh).astype(int)
print(classification_report(y_test, y_pred_tuned, digits=4))
```

```
Chosen threshold for recall>=0.90 -> 0.0000
              precision    recall  f1-score   support

           0     0.0000    0.0000    0.0000     71079
           1     0.0017    1.0000    0.0034       123

    accuracy                         0.0017     71202
   macro avg     0.0009    0.5000    0.0017     71202
weighted avg     0.0000    0.0017    0.0000     71202

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defi
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defi
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is ill-defi
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
import shap
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings("ignore")

# --- 0) safety: small background sample & sample for explanation
background = X_train.sample(n=min(200, len(X_train)), random_state=RANDOM_STATE)
X_sample = X_test.sample(n=min(200, len(X_test)), random_state=RANDOM_STATE)

# --- 1) create callable that returns prob of class 1
def model_predict_proba_positive(X):
    # SHAP may pass a numpy array — convert to DataFrame with correct columns
    if not isinstance(X, pd.DataFrame):
        X = pd.DataFrame(X, columns=background.columns)
    return pipe.predict_proba(X)[:, 1]

# --- 2) Try the modern/simple approach: shap.Explainer(callable, background)
try:
    explainer = shap.Explainer(model_predict_proba_positive, background)
    shap_vals = explainer(X_sample)   # new API object
    print("Used shap.Explainer successfully (new API).")
    # Plots (new API)
    print("Rendering global summary (beeswarm)...")
    shap.plots.beeswarm(shap_vals)       # global
    # Waterfall for first sample
    print("Rendering waterfall for first sample...")
```

```
        shap.plots.waterfall(shap_vals[0])
    except Exception as e:
        print("shap.Explainer failed with:", repr(e))
        print("Falling back to KernelExplainer (slower).")

        # --- 3) Fallback: KernelExplainer (model-agnostic, but slow)
        # Use extremely small background for KernelExplainer to avoid long runtime
        ker_background = background.sample(n=min(50, len(background)), random_state=RANDOM_STATE)
        # KernelExplainer expects a function returning predictions for matrix input
        try:
            ke = shap.KernelExplainer(model_predict_proba_positive, ker_background)
            # compute shap values for a very small subset (e.g., 20 samples max)
            X_small = X_sample.sample(n=min(20, len(X_sample)), random_state=RANDOM_STATE)
            shap_vals_ker = ke.shap_values(X_small, nsamples=100)  # nsamples tradeoff speed/accuracy
            print("KernelExplainer succeeded. Plotting summary for small subset...")
            shap.summary_plot(shap_vals_ker, X_small)
        except Exception as e2:
            print("KernelExplainer also failed with:", repr(e2))
            print("As last resort, please tell me your shap.__version__ and xgboost.__version__ and I'll give version-specific co
```

```
PermutationExplainer explainer: 201it [01:31,  2.12it/s]
Used shap.Explainer successfully (new API).
Rendering global summary (beeswarm)...
```



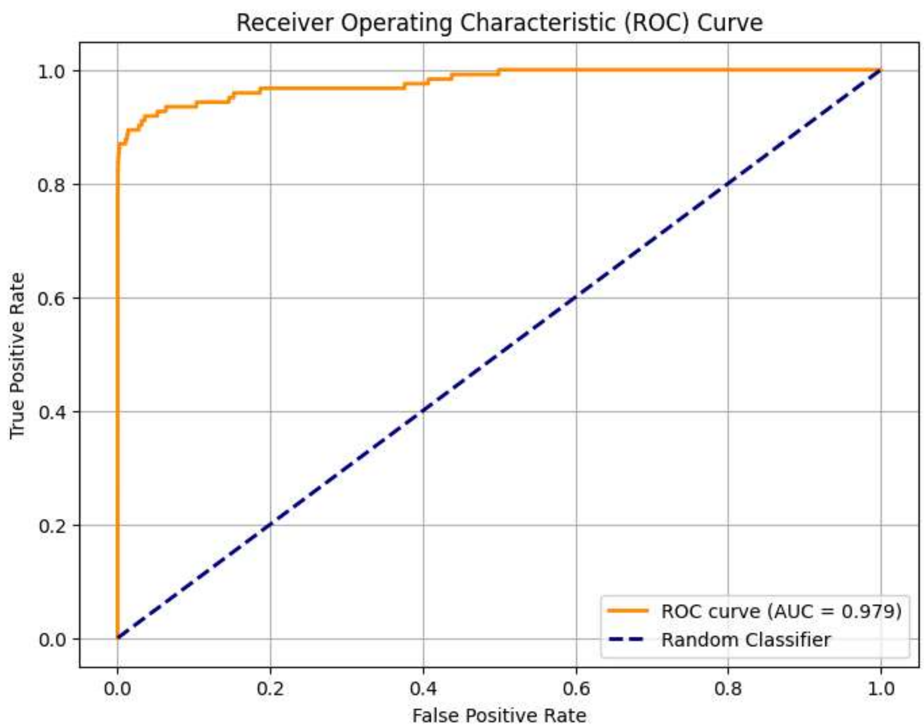

```
Rendering waterfall for first sample...
```


```
param_grid = {
    'clf__n_estimators': [100, 200],
    'clf__max_depth': [4, 6],
    'clf__learning_rate': [0.05, 0.1],
}

grid = GridSearchCV(pipe, param_grid, scoring='average_precision', cv=3, n_jobs=-1, verbose=2)
grid.fit(X_train, y_train)
print("Best params:", grid.best_params_)
best_pipe = grid.best_estimator_

# Evaluate best
y_proba_grid = best_pipe.predict_proba(X_test)[:,1]
```

```
prec, rec, _ = precision_recall_curve(y_test, y_proba_grid)
print("Tuned PR AUC:", auc(rec, prec))
```

```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
Best params: {'clf__learning_rate': 0.1, 'clf__max_depth': 4, 'clf__n_estimators': 200}
Tuned PR AUC: 0.8402749671242524
```

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Compute ROC curve and ROC area
fpr, tpr, thresholds = roc_curve(y_test, y_proba_grid)
roc_auc = auc(fpr, tpr)

# Plot
plt.figure(figsize=(8,6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.3f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label='Random Classifier')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```




```
import pandas as pd
# sample some thresholds (avoid first and last)
df_pr = pd.DataFrame({'precision': prec[:-1], 'recall': rec[:-1], 'threshold': thresh})
# show thresholds with recall >= [0.9, 0.8, 0.7] etc.
```