```
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, cross_val_score,
GridSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    roc_auc_score, confusion_matrix, classification_report, roc_curve
)
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
import matplotlib.pyplot as plt

RANDOM_STATE = 42

import os

DATA_PATH = 'credit_data.csv'  # change if you have a local file

if os.path.exists(DATA_PATH):
    df = pd.read_csv(DATA_PATH)
    print(f"Loaded data from {DATA_PATH} with shape {df.shape}")
else:
    # Create synthetic dataset with realistic-ish features
    from sklearn.datasets import make_classification

    X, y = make_classification(
        n_samples=5000,
        n_features=12,
        n_informative=8,
        n_redundant=2,
        n_clusters_per_class=2,
        weights=[0.7, 0.3],  # class imbalance: more 'good' credit
        flip_y=0.03,
        random_state=RANDOM_STATE
    )

    df = pd.DataFrame(X, columns=[
        'income', 'debt', 'credit_utilization', 'age', 'loan_amount',
        'num_credit_lines', 'num_open_accounts', 'years_on_job',
'savings',
        'num_derogatory_marks', 'home_ownership_flag', 'other_feature'
```

```
    ])
    df['employment_status'] = np.random.choice(
        ['employed', 'self-employed', 'unemployed', 'retired'],
size=len(df), p=[0.7,0.15,0.1,0.05]
    )
     # Target Variable: Creditworthiness = {Good Credit (1), Bad
Credit (0)}
    df['target'] = y  # 1 -> Good Credit, 0 -> Bad Credit
    # Introduce some missing values randomly
    for col in ['income', 'debt', 'years_on_job', 'savings']:
        df.loc[df.sample(frac=0.05, random_state=RANDOM_STATE).index,
col] = np.nan

    print("Synthetic dataset created with shape:", df.shape)

# Quick preview
print(df.head())
```

```
Synthetic dataset created with shape: (5000, 14)
     income      debt  credit_utilization       age  loan_amount  \
0 -0.246718  3.363401           -0.564936  0.820274     3.083007
1  3.497554  3.300090            1.659837  0.154689     1.151323
2  1.323520  2.144484            1.001996 -0.570958    -1.456459
3 -1.015198  0.321040           -2.331566 -0.524955     1.368296
4 -1.675479  0.742785           -0.840219  1.385035    -0.426183

   num_credit_lines  num_open_accounts  years_on_job   savings  \
0          2.042685           0.125531      1.908001  1.195192
1          4.492193          -1.733373      2.431870 -1.153688
2          2.753598           2.260532      1.850505 -0.843093
3         -2.302294           0.798896      3.437645 -0.302643
4         -1.378756          -1.531374      0.378909 -0.587425

   num_derogatory_marks  home_ownership_flag  other_feature
employment_status  \
0              1.808655            -1.139616       0.864853
employed
1              1.022063             1.199785      -3.508381
employed
2              1.090812             0.565385       1.120241
employed
3             -4.152464             0.924537       4.325814     self-
employed
4             -1.787170            -0.945137       0.812472
employed

   target
0       1
1       0
2       0
```

```
3        0
4        0

TARGET = 'target'

# Heuristically group columns
numeric_features =
df.select_dtypes(include=[np.number]).columns.tolist()
# Remove the target from numeric_features if present
if TARGET in numeric_features:
    numeric_features.remove(TARGET)

categorical_features = df.select_dtypes(include=['object',
'category']).columns.tolist()

print('Numeric features:', numeric_features)
print('Categorical features:', categorical_features)

Numeric features: ['income', 'debt', 'credit_utilization', 'age',
'loan_amount', 'num_credit_lines', 'num_open_accounts',
'years_on_job', 'savings', 'num_derogatory_marks',
'home_ownership_flag', 'other_feature']
Categorical features: ['employment_status']

numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore', drop='first'))
])

preprocessor = ColumnTransformer(transformers=[
    ('num', numeric_transformer, numeric_features),
    ('cat', categorical_transformer, categorical_features)
])

X = df.drop(columns=[TARGET])
y = df[TARGET]

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=RANDOM_STATE, stratify=y
)

print('Train shape:', X_train.shape, 'Test shape:', X_test.shape)

# %%
"""
Cell 6: Model helper to train and evaluate models
```

```
"""
def evaluate_model(model, X_test, y_test, model_name='Model'):
    y_pred = model.predict(X_test)
    y_proba = None
    if hasattr(model, 'predict_proba'):
        y_proba = model.predict_proba(X_test)[:, 1]
    elif hasattr(model, 'decision_function'):
        # some models like SVM may have decision_function
        y_proba = model.decision_function(X_test)

    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred)
    rec = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_proba) if y_proba is not None
else None

    print(f"--- {model_name} Evaluation ---")
    print('Accuracy:', round(acc, 4))
    print('Precision:', round(prec, 4))
    print('Recall:', round(rec, 4))
    print('F1-score:', round(f1, 4))
    if roc_auc is not None:
     print('ROC-AUC:', round(roc_auc, 4))
    print('\nConfusion Matrix (rows = Actual, cols = Predicted):')
    print(pd.DataFrame(confusion_matrix(y_test, y_pred),
                       index=['Actual Bad Credit (0)', 'Actual Good
Credit (1)'],
                       columns=['Predicted Bad Credit (0)', 'Predicted
Good Credit (1)']))
    print('\nClassification Report:')
    print(classification_report(y_test, y_pred, target_names=['Bad
Credit (0)', 'Good Credit (1)']))

    return {'accuracy': acc, 'precision': prec, 'recall': rec, 'f1':
f1, 'roc_auc': roc_auc}

Train shape: (4000, 13) Test shape: (1000, 13)

log_clf = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('clf', LogisticRegression(random_state=RANDOM_STATE,
max_iter=1000))
])

log_clf.fit(X_train, y_train)
metrics_log = evaluate_model(log_clf, X_test, y_test,
model_name='Logistic Regression')
```

```
--- Logistic Regression Evaluation ---
Accuracy: 0.845
Precision: 0.8333
Recall: 0.6106
F1-score: 0.7048
ROC-AUC: 0.8298

Confusion Matrix (rows = Actual, cols = Predicted):
                        Predicted Bad Credit (0)  Predicted Good
Credit (1)
Actual Bad Credit (0)                        660
37
Actual Good Credit (1)                       118
185

Classification Report:
                 precision    recall  f1-score   support

 Bad Credit (0)       0.85      0.95      0.89       697
Good Credit (1)       0.83      0.61      0.70       303

       accuracy                           0.84      1000
      macro avg       0.84      0.78      0.80      1000
   weighted avg       0.84      0.84      0.84      1000
```

```
dt_clf = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('clf', DecisionTreeClassifier(random_state=RANDOM_STATE))
])

dt_clf.fit(X_train, y_train)
metrics_dt = evaluate_model(dt_clf, X_test, y_test,
model_name='Decision Tree')
```

```
--- Decision Tree Evaluation ---
Accuracy: 0.878
Precision: 0.8007
Recall: 0.7954
F1-score: 0.798
ROC-AUC: 0.8546

Confusion Matrix (rows = Actual, cols = Predicted):
                        Predicted Bad Credit (0)  Predicted Good
Credit (1)
Actual Bad Credit (0)                        637
60
Actual Good Credit (1)                        62
241
```

```
Classification Report:
                 precision    recall  f1-score   support

 Bad Credit (0)       0.91      0.91      0.91       697
Good Credit (1)       0.80      0.80      0.80       303

       accuracy                           0.88      1000
      macro avg       0.86      0.85      0.86      1000
   weighted avg       0.88      0.88      0.88      1000
```

```
rf_clf = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('clf', RandomForestClassifier(random_state=RANDOM_STATE, n_jobs=-
1))
])

rf_clf.fit(X_train, y_train)
metrics_rf = evaluate_model(rf_clf, X_test, y_test, model_name='Random
Forest')
```

```
--- Random Forest Evaluation ---
Accuracy: 0.938
Precision: 0.9513
Recall: 0.8383
F1-score: 0.8912
ROC-AUC: 0.9796

Confusion Matrix (rows = Actual, cols = Predicted):
                        Predicted Bad Credit (0)  Predicted Good
Credit (1)
Actual Bad Credit (0)                        684
13
Actual Good Credit (1)                        49
254

Classification Report:
                 precision    recall  f1-score   support

 Bad Credit (0)       0.93      0.98      0.96       697
Good Credit (1)       0.95      0.84      0.89       303

       accuracy                           0.94      1000
      macro avg       0.94      0.91      0.92      1000
   weighted avg       0.94      0.94      0.94      1000
```

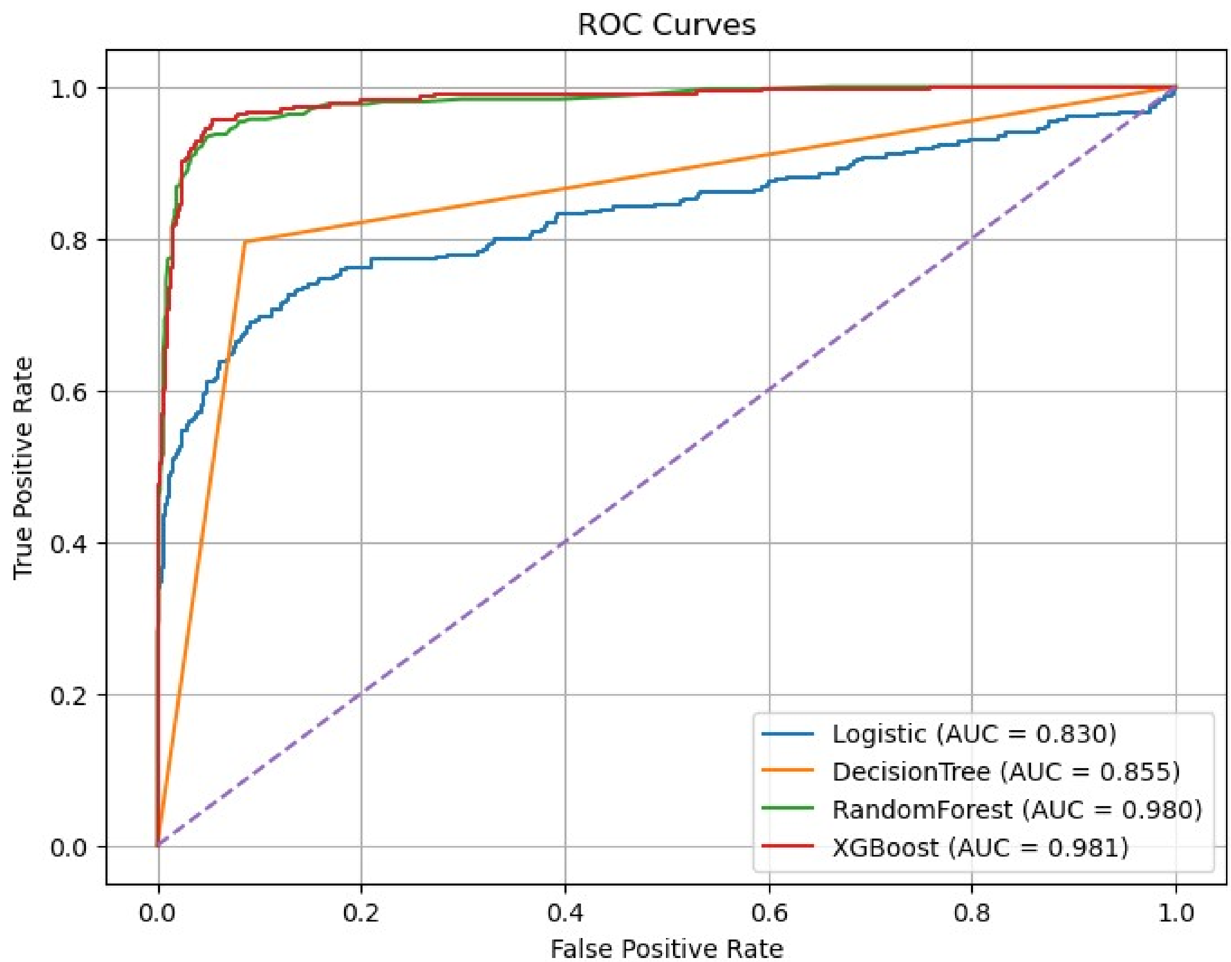
```
xgb_clf = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('clf', xgb.XGBClassifier(use_label_encoder=False,
eval_metric='logloss', random_state=RANDOM_STATE, n_jobs=4))
```

```
])

xgb_clf.fit(X_train, y_train)
metrics_xgb = evaluate_model(xgb_clf, X_test, y_test,
model_name='XGBoost')

# %%
"""
Cell 11: Compare models (simple summary)
"""
results = pd.DataFrame([
    {**metrics_log, 'model': 'LogisticRegression'},
    {**metrics_dt, 'model': 'DecisionTree'},
    {**metrics_rf, 'model': 'RandomForest'},
    {**metrics_xgb, 'model': 'XGBoost'}
])

results = results[['model', 'accuracy', 'precision', 'recall', 'f1',
'roc_auc']]
print(results.sort_values('roc_auc', ascending=False))
```

```
--- XGBoost Evaluation ---
Accuracy: 0.952
Precision: 0.9412
Recall: 0.8977
F1-score: 0.9189
ROC-AUC: 0.9813

Confusion Matrix (rows = Actual, cols = Predicted):
                        Predicted Bad Credit (0)  Predicted Good
Credit (1)
Actual Bad Credit (0)                        680
17
Actual Good Credit (1)                        31
272

Classification Report:
                 precision    recall  f1-score   support

 Bad Credit (0)       0.96      0.98      0.97       697
Good Credit (1)       0.94      0.90      0.92       303

       accuracy                           0.95      1000
      macro avg       0.95      0.94      0.94      1000
   weighted avg       0.95      0.95      0.95      1000

                model  accuracy  precision    recall        f1
roc_auc
3             XGBoost     0.952   0.941176  0.897690  0.918919
0.981282
```

```
2         RandomForest     0.938   0.951311  0.838284  0.891228
0.979618
1         DecisionTree     0.878   0.800664  0.795380  0.798013
0.854648
0  LogisticRegression     0.845   0.833333  0.610561  0.704762
0.829765
```

```
plt.figure(figsize=(8,6))
for pipe, name in [(log_clf, 'Logistic'), (dt_clf, 'DecisionTree'),
(rf_clf, 'RandomForest'), (xgb_clf, 'XGBoost')]:
    if hasattr(pipe, 'predict_proba'):
        y_score = pipe.predict_proba(X_test)[:, 1]
    else:
        try:
            y_score = pipe.decision_function(X_test)
        except Exception:
            continue
    fpr, tpr, _ = roc_curve(y_test, y_score)
    auc = roc_auc_score(y_test, y_score)
    plt.plot(fpr, tpr, label=f"{name} (AUC = {auc:.3f})")

plt.plot([0,1], [0,1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves')
plt.legend()
plt.grid(True)
plt.show()
```




```
# Fit preprocessor separately to get transformed feature names
preprocessor.fit(X_train)

# Get numeric feature names (after scaling they keep same names)
num_features_after = numeric_features

# Get categorical feature names after one-hot encoding
cat_encoder =
preprocessor.named_transformers_['cat'].named_steps['onehot']
cat_feature_names = []
if hasattr(cat_encoder, 'get_feature_names_out'):
    cat_feature_names =
cat_encoder.get_feature_names_out(categorical_features).tolist()

feature_names = num_features_after + cat_feature_names

# Random Forest feature importances
rf_clf.named_steps['clf'].feature_importances_
rf_importances = rf_clf.named_steps['clf'].feature_importances_
rf_imp_df = pd.DataFrame({'feature': feature_names, 'importance':
```

```
rf_importances}).sort_values('importance', ascending=False).head(20)

print('Top Random Forest features:')
print(rf_imp_df)

# XGBoost feature importances
xgb_importances = xgb_clf.named_steps['clf'].feature_importances_
xgb_imp_df = pd.DataFrame({'feature': feature_names, 'importance':
xgb_importances}).sort_values('importance', ascending=False).head(20)

print('\nTop XGBoost features:')
print(xgb_imp_df)
```

```
Top Random Forest features:
                           feature  importance
8                          savings    0.135916
6                num_open_accounts    0.130648
11                   other_feature    0.123415
5                 num_credit_lines    0.123356
7                     years_on_job    0.085483
0                           income    0.081301
1                             debt    0.074619
9             num_derogatory_marks    0.069601
2               credit_utilization    0.055914
4                      loan_amount    0.053931
10             home_ownership_flag    0.029093
3                              age    0.028632
13  employment_status_self-employed    0.003397
12        employment_status_retired    0.002528
14     employment_status_unemployed    0.002165

Top XGBoost features:
                           feature  importance
8                          savings    0.154237
11                   other_feature    0.122278
6                num_open_accounts    0.112575
5                 num_credit_lines    0.108904
1                             debt    0.084809
0                           income    0.081974
7                     years_on_job    0.081734
9             num_derogatory_marks    0.063649
2               credit_utilization    0.048141
4                      loan_amount    0.045043
14     employment_status_unemployed    0.024824
10             home_ownership_flag    0.019248
3                              age    0.018552
12        employment_status_retired    0.018028
13  employment_status_self-employed    0.016004
```

```
best_model_name = results.sort_values('roc_auc',
ascending=False).iloc[0]['model']
print('Best model according to ROC-AUC:', best_model_name)

best_pipeline = {
    'LogisticRegression': log_clf,
    'DecisionTree': dt_clf,
    'RandomForest': rf_clf,
    'XGBoost': xgb_clf
}[best_model_name]

import joblib
joblib.dump(best_pipeline, 'best_credit_model.joblib')
print('Saved best model to best_credit_model.joblib')
```

```
Best model according to ROC-AUC: XGBoost
Saved best model to best_credit_model.joblib
```