In [ ]:

In [2]:
```
import pandas as pd
import numpy as np
from pathlib import Path

# 基础路径
base_path = Path(r"C:\Users\田\Desktop\python实操\kaggle\Credit Card Approval Prediction")  # 你的原始路径
```

In [1]:
```
import toad
import scorecardpy as sc
```

In [3]:
```
application_record = pd.read_csv(base_path / 'application_record.csv')
credit_record = pd.read_csv(base_path / 'credit_record.csv')
```

In [ ]:

In [4]:
```
credit_record
```

Out[4]:

	ID	MONTHS_BALANCE	STATUS
0	5001711	0	X
1	5001711	-1	0
2	5001711	-2	0
3	5001711	-3	0
4	5001712	0	C
...	...	...	...
1048570	5150487	-25	C
1048571	5150487	-26	C
1048572	5150487	-27	C
1048573	5150487	-28	C
1048574	5150487	-29	C

1048575 rows × 3 columns

In [6]:
```
credit_record.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 3 columns):
 #   Column          Non-Null Count    Dtype
---  ------          --------------    -----
 0   ID              1048575 non-null  int64
 1   MONTHS_BALANCE  1048575 non-null  int64
 2   STATUS          1048575 non-null  object
dtypes: int64(2), object(1)
memory usage: 24.0+ MB
```

In [7]:
```
credit_record.describe()
```

Out[7]:

	ID	MONTHS_BALANCE
count	1.048575e+06	1.048575e+06
mean	5.068286e+06	-1.913700e+01
std	4.615058e+04	1.402350e+01
min	5.001711e+06	-6.000000e+01
25%	5.023644e+06	-2.900000e+01
50%	5.062104e+06	-1.700000e+01
75%	5.113856e+06	-7.000000e+00
max	5.150487e+06	0.000000e+00

In [11]:
```
credit_record.describe(include='object')
```

Out[11]:

	STATUS
count	1048575
unique	8
top	C
freq	442031

In [ ]:
```

```

In [5]:
```
application_record
```

Out[5]:

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE	NAME_I
0	5008804	M	Y	Y	0	427500.0	Working	Higher education	
1	5008805	M	Y	Y	0	427500.0	Working	Higher education	
2	5008806	M	Y	Y	0	112500.0	Working	Secondary / secondary special	
3	5008808	F	N	Y	0	270000.0	Commercial associate	Secondary / secondary special	Sing
4	5008809	F	N	Y	0	270000.0	Commercial associate	Secondary / secondary special	Sing
...	...	...	...	...	...	...	...	...	
438552	6840104	M	N	Y	0	135000.0	Pensioner	Secondary / secondary special	
438553	6840222	F	N	N	0	103500.0	Working	Secondary / secondary special	Sing
438554	6841878	F	N	N	0	54000.0	Commercial associate	Higher education	Sing
438555	6842765	F	N	Y	0	72000.0	Pensioner	Secondary / secondary special	
438556	6842885	F	N	Y	0	121500.0	Working	Secondary / secondary special	

438557 rows × 18 columns

In [8]:
```
application_record.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 438557 entries, 0 to 438556
Data columns (total 18 columns):
 #   Column               Non-Null Count   Dtype  
---  ------               --------------   -----  
 0   ID                   438557 non-null  int64  
 1   CODE_GENDER          438557 non-null  object 
 2   FLAG_OWN_CAR         438557 non-null  object 
 3   FLAG_OWN_REALTY      438557 non-null  object 
 4   CNT_CHILDREN         438557 non-null  int64  
 5   AMT_INCOME_TOTAL     438557 non-null  float64
 6   NAME_INCOME_TYPE     438557 non-null  object 
 7   NAME_EDUCATION_TYPE  438557 non-null  object 
 8   NAME_FAMILY_STATUS   438557 non-null  object 
 9   NAME_HOUSING_TYPE    438557 non-null  object 
 10  DAYS_BIRTH           438557 non-null  int64  
 11  DAYS_EMPLOYED        438557 non-null  int64  
 12  FLAG_MOBIL           438557 non-null  int64  
 13  FLAG_WORK_PHONE      438557 non-null  int64  
 14  FLAG_PHONE           438557 non-null  int64  
 15  FLAG_EMAIL           438557 non-null  int64  
 16  OCCUPATION_TYPE      304354 non-null  object 
 17  CNT_FAM_MEMBERS      438557 non-null  float64
dtypes: float64(2), int64(8), object(8)
memory usage: 60.2+ MB
```

In [9]:
```
application_record.describe()
```

Out[9]:

	ID	CNT_CHILDREN	AMT_INCOME_TOTAL	DAYS_BIRTH	DAYS_EMPLOYED	FLAG_MOBIL	FLAG_WORK_PHONE	FLAG_PHONE	FLAG_EMAIL	CNT_F
count	4.385570e+05	438557.000000	4.385570e+05	438557.000000	438557.000000	438557.0	438557.000000	438557.000000	438557.000000	
mean	6.022176e+06	0.427390	1.875243e+05	-15997.904649	60563.675328	1.0	0.206133	0.287771	0.108207	
std	5.716370e+05	0.724882	1.100869e+05	4185.030007	138767.799647	0.0	0.404527	0.452724	0.310642	
min	5.008804e+06	0.000000	2.610000e+04	-25201.000000	-17531.000000	1.0	0.000000	0.000000	0.000000	
25%	5.609375e+06	0.000000	1.215000e+05	-19483.000000	-3103.000000	1.0	0.000000	0.000000	0.000000	
50%	6.047745e+06	0.000000	1.607805e+05	-15630.000000	-1467.000000	1.0	0.000000	0.000000	0.000000	
75%	6.456971e+06	1.000000	2.250000e+05	-12514.000000	-371.000000	1.0	0.000000	1.000000	0.000000	
max	7.999952e+06	19.000000	6.750000e+06	-7489.000000	365243.000000	1.0	1.000000	1.000000	1.000000	

In [10]:
```
application_record.describe(include='object')
```

Out[10]:

	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE	NAME_FAMILY_STATUS	NAME_HOUSING_TYPE	OCCUPAT
count	438557	438557	438557	438557	438557	438557	438557	
unique	2	2	2	5	5	5	6	
top	F	N	Y	Working	Secondary / secondary special	Married	House / apartment	
freq	294440	275459	304074	226104	301821	299828	393831	

In [28]:
```
credit_data['STATUS'].value_counts()
```

Out[28]:
```
STATUS
C    442031
0    383120
X    209230
1     11090
5      1693
2       868
3       320
4       223
Name: count, dtype: int64
```

In [ ]:

In [ ]:

In [ ]:

TOAD评分卡建模

In [27]:
```
# ==================== TOAD评分卡建模 ====================
# 文件名: credit_scoring_toad.py

import toad
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             f1_score, roc_auc_score, confusion_matrix, roc_curve)
from scipy.stats import ks_2samp
import warnings
warnings.filterwarnings('ignore')

# ==================== 1. 数据加载 ====================
```

```
print("="*60)
print("TOAD评分卡建模")
print("="*60)

# 加载数据
app_data = application_record
credit_data = credit_record

print(f"申请数据: {app_data.shape}, 信用记录: {credit_data.shape}")

# ==================== 2. 标签构建 ====================
def create_target_label(status):
    """创建目标标签"""
    if status in ['C', 'X', '0']:
        return 1  # Good
    else:
        return 0  # Bad

credit_data['label'] = credit_data['STATUS'].apply(create_target_label)

# 聚合每个客户ID的标签（多数投票）
customer_label = credit_data.groupby('ID')['label'].agg(lambda x: 1 if x.mean() >= 0.5 else 0).reset_index()
customer_label.columns = ['ID', 'label']

print(f"\n标签分布:")
print(f"Good(1): {(customer_label['label']==1).sum():,} ({(customer_label['label']==1).sum()/len(customer_label)*100:.1f}%)")
print(f"Bad(0):  {(customer_label['label']==0).sum():,} ({(customer_label['label']==0).sum()/len(customer_label)*100:.1f}%)")

# ==================== 3. 数据合并与预处理 ====================
# 填充缺失值
app_data['OCCUPATION_TYPE'] = app_data['OCCUPATION_TYPE'].fillna('Unknown')

# 合并数据
data = pd.merge(app_data, customer_label, on='ID', how='inner')
print(f"\n合并后数据: {data.shape}")

# 移除ID列
data = data.drop('ID', axis=1)

# ==================== 4. TOAD特征筛选 ====================
print("\n" + "="*60)
print("TOAD特征筛选")
print("="*60)

# 使用TOAD进行特征筛选
selected_result = toad.selection.select(
    data,
    target='label',
    empty=0.6,      # 删除缺失率>60%的特征
    iv=0.02,        # 保留IV>0.02的特征
    corr=0.7,       # 删除相关性>0.7的冗余特征
```

```
    return_drop=True
)

# 提取筛选后的数据
if isinstance(selected_result, tuple):
    selected_data = selected_result[0]
    dropped_features = selected_result[1]
    print(f"特征筛选结果: {data.shape[1]} -> {selected_data.shape[1]} 个特征")
    print(f"\n被删除特征统计:")
    for reason, features in dropped_features.items():
        if features is not None and len(features) > 0:  # ✅
            print(f"  {reason}: {len(features)}个特征")

        # if features:
        #     print(f"  {reason}: {len(features)}个特征")
else:
    selected_data = selected_result

# ==================== 5. 数据划分 ====================
print("\n" + "="*60)
print("数据划分")
print("="*60)

# 划分训练测试集
train, test = train_test_split(selected_data, test_size=0.3, random_state=42, stratify=selected_data['label'])

print(f"训练集: {train.shape}")
print(f"测试集: {test.shape}")
print(f"\n训练集标签分布:")
print(train['label'].value_counts())
print(f"\n测试集标签分布:")
print(test['label'].value_counts())

# ==================== 6. TOAD分箱处理 ====================
print("\n" + "="*60)
print("TOAD分箱处理")
print("="*60)

# 初始化分箱器
combiner = toad.transform.Combiner()

# 使用卡方分箱
combiner.fit(train.drop('label', axis=1), y=train['label'],
             method='chi', min_samples=0.05, n_bins=8, empty_separate=True)

# 应用分箱
train_binned = combiner.transform(train.drop('label', axis=1))
test_binned = combiner.transform(test.drop('label', axis=1))

print(f"分箱完成，特征数: {train_binned.shape[1]}")
```

```python
# ==================== 7. WOE转换 ====================
print("\n" + "="*60)
print("WOE转换")
print("="*60)

# 初始化WOE转换器
transformer = toad.transform.WOETransformer()

# 拟合并转换训练集
train_woe = transformer.fit_transform(train_binned, train['label'])
test_woe = transformer.transform(test_binned)

print(f"训练集WOE转换后: {train_woe.shape}")
print(f"测试集WOE转换后: {test_woe.shape}")

# 计算IV值
train_woe['label'] = train['label'].values
test_woe['label'] = test['label'].values

# iv_scores = toad.quality(train_woe, 'label', iv_only=True)
iv_scores = toad.quality(train_woe, 'label', iv_only=True, cpu_cores=1)
print(f"\n特征IV值统计:")
print(f"IV > 0.3 (强预测力): {(iv_scores > 0.3).sum()} 个")
print(f"IV 0.1-0.3 (中等预测力): {((iv_scores >= 0.1) & (iv_scores <= 0.3)).sum()} 个")
print(f"IV 0.02-0.1 (弱预测力): {((iv_scores >= 0.02) & (iv_scores < 0.1)).sum()} 个")
print(f"IV < 0.02 (无预测力): {(iv_scores < 0.02).sum()} 个")

# ==================== 8. 逻辑回归建模 ====================
print("\n" + "="*60)
print("逻辑回归建模")
print("="*60)

# 准备数据
X_train = train_woe
y_train = train['label']
X_test = test_woe
y_test = test['label']

# 检查数据
if np.any(np.isinf(X_train)) or np.any(np.isinf(X_test)):
    X_train = X_train.replace([np.inf, -np.inf], np.nan)
    X_test = X_test.replace([np.inf, -np.inf], np.nan)
    X_train = X_train.fillna(0)
    X_test = X_test.fillna(0)

# 训练逻辑回归
lr = LogisticRegression(C=0.1, max_iter=1000, random_state=42, solver='lbfgs')
lr.fit(X_train, y_train)

# 预测
y_pred = lr.predict(X_test)
```

```python
y_pred_proba = lr.predict_proba(X_test)[:, 1]

# ==================== 9. 模型评估 ====================
print("\n" + "="*60)
print("模型评估")
print("="*60)

# 计算基础指标
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, zero_division=0)
recall = recall_score(y_test, y_pred, zero_division=0)
f1 = f1_score(y_test, y_pred, zero_division=0)
roc_auc = roc_auc_score(y_test, y_pred_proba)

print(f"准确率: {accuracy:.4f}")
print(f"精确率: {precision:.4f}")
print(f"召回率: {recall:.4f}")
print(f"F1分数: {f1:.4f}")
print(f"ROC-AUC: {roc_auc:.4f}")

# 计算KS值
def calculate_ks(y_true, y_pred_proba):
    good_proba = y_pred_proba[y_true == 1]
    bad_proba = y_pred_proba[y_true == 0]
    if len(good_proba) > 0 and len(bad_proba) > 0:
        ks_stat, _ = ks_2samp(good_proba, bad_proba)
        return ks_stat
    return 0

ks_value = calculate_ks(y_test, y_pred_proba)
print(f"KS值: {ks_value:.4f}")

# 混淆矩阵
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()
print(f"\n混淆矩阵:")
print(f"TN(正确拒绝坏客户): {tn}")
print(f"FP(错误批准坏客户): {fp}")
print(f"FN(错误拒绝好客户): {fn}")
print(f"TP(正确批准好客户): {tp}")

# 计算误分类成本
bad_loss = 3000  # 坏客户造成的平均损失
good_revenue = 300  # 好客户带来的平均收益
cost = fp * bad_loss + fn * good_revenue
print(f"\n误分类成本: ${cost:,.0f}")

# ==================== 10. 生成评分卡 ====================
print("\n" + "="*60)
print("生成评分卡")
print("="*60)
```

```python
# 使用TOAD生成评分卡
# 基准参数
P0 = 600  # 基准分
PDO = 20  # 分数翻倍所需odds变化
odds0 = 1/30  # 基准好坏比

# 计算系数
B = PDO / np.log(2)
A = P0 - B * np.log(odds0)

# 获取特征系数
feature_names = X_train.columns
coefficients = lr.coef_[0]
intercept = lr.intercept_[0]


# 计算基础分
base_score = round(A - B * intercept)
print(f"基础分: {base_score}")

# ==================== 11. 计算客户分数 ====================
print("\n" + "="*60)
print("计算客户分数")
print("="*60)

# 计算训练集和测试集分数
def calculate_score(row, scorecard, base_score, transformer):
    score = base_score
    for feature in scorecard.keys():
        if feature in row.index:
            bin_value = row[feature]
            if bin_value in scorecard[feature]:
                score += scorecard[feature][bin_value]
    return score

train_scores = train_binned.apply(lambda row: calculate_score(row, scorecard, base_score, transformer), axis=1)
test_scores = test_binned.apply(lambda row: calculate_score(row, scorecard, base_score, transformer), axis=1)

print(f"训练集分数范围: {train_scores.min():.0f} - {train_scores.max():.0f}")
print(f"训练集平均分: {train_scores.mean():.0f}")
print(f"测试集分数范围: {test_scores.min():.0f} - {test_scores.max():.0f}")
print(f"测试集平均分: {test_scores.mean():.0f}")

# 计算PSI
psi = toad.metrics.PSI(train_scores, test_scores)
print(f"PSI值: {psi:.4f}")

# ==================== 12. 保存结果 ====================
print("\n" + "="*60)
print("保存结果")
```

```python
print("="*60)

# 保存评分卡
scorecard_df = pd.DataFrame([
    {
        'feature': feature,
        'bin': str(bin_value),
        'woe': transformer.woe_dict.get(feature, {}).get(bin_value, 0),
        'score': score,
        'coefficient': coefficients[list(feature_names).index(feature)] if feature in feature_names else 0
    }
    for feature, bin_scores in scorecard.items()
    for bin_value, score in bin_scores.items()
])

scorecard_df.to_csv('toad_scorecard.csv', index=False)
print(f"✓ TOAD评分卡已保存: toad_scorecard.csv")

# 保存预测结果
test_result = test.copy()
test_result['predicted_label'] = y_pred
test_result['predicted_prob'] = y_pred_proba
test_result['score'] = test_scores.values
test_result.to_csv('toad_test_predictions.csv', index=False)
print(f"✓ 测试集预测结果已保存: toad_test_predictions.csv")

# 保存模型系数
coef_df = pd.DataFrame({
    'feature': feature_names,
    'coefficient': coefficients
}).sort_values('coefficient', key=abs, ascending=False)
coef_df.to_csv('toad_model_coefficients.csv', index=False)
print(f"✓ 模型系数已保存: toad_model_coefficients.csv")


print("\n" + "="*60)
print("TOAD评分卡建模完成")
print("="*60)
print(f"总特征数: {selected_data.shape[1] - 1}")
print(f"模型AUC: {roc_auc:.4f}")
print(f"模型KS: {ks_value:.4f}")
print(f"模型PSI: {psi:.4f}")
```

```
============================================================
TOAD评分卡建模
============================================================
申请数据: (438557, 18), 信用记录: (1048575, 4)

标签分布:
Good(1): 45,847 (99.7%)
Bad(0):  138 (0.3%)

合并后数据: (36457, 19)

============================================================
TOAD特征筛选
============================================================
特征筛选结果: 18 -> 12 个特征

被删除特征统计:
  iv: 5个特征
  corr: 1个特征

============================================================
数据划分
============================================================
训练集: (25519, 12)
测试集: (10938, 12)

训练集标签分布:
label
1    25433
0       86
Name: count, dtype: int64

测试集标签分布:
label
1    10901
0       37
Name: count, dtype: int64

============================================================
TOAD分箱处理
============================================================
分箱完成，特征数: 11

============================================================
WOE转换
============================================================
训练集WOE转换后: (25519, 11)
测试集WOE转换后: (10938, 11)

特征IV值统计:
IV > 0.3 (强预测力): iv          2
```

```
gini        0
entropy     0
unique     11
dtype: int64 个
IV 0.1-0.3 (中等预测力): iv         4
gini       0
entropy    0
unique     0
dtype: int64 个
IV 0.02-0.1 (弱预测力): iv         5
gini       0
entropy    0
unique     0
dtype: int64 个
IV < 0.02 (无预测力): iv         0
gini       0
entropy    0
unique     0
dtype: int64 个

============================================================
逻辑回归建模
============================================================

============================================================
模型评估
============================================================
准确率: 0.9969
精确率: 0.9969
召回率: 1.0000
F1分数: 0.9984
ROC-AUC: 1.0000
KS值: 1.0000

混淆矩阵:
TN(正确拒绝坏客户): 3
FP(错误批准坏客户): 34
FN(错误拒绝好客户): 0
TP(正确批准好客户): 10901

误分类成本: $102,000

============================================================
生成评分卡
============================================================
基础分: 673

============================================================
计算客户分数
============================================================
训练集分数范围: 673 - 673
```

```
训练集平均分: 673
测试集分数范围: 673 - 673
测试集平均分: 673
PSI值: 0.0000

============================================================
保存结果
============================================================
✓ TOAD评分卡已保存: toad_scorecard.csv
✓ 测试集预测结果已保存: toad_test_predictions.csv
✓ 模型系数已保存: toad_model_coefficients.csv

============================================================
TOAD评分卡建模完成
============================================================
总特征数: 11
模型AUC: 1.0000
模型KS: 1.0000
模型PSI: 0.0000
```

In [ ]:

Scorecardpy评分卡建模

In [26]:

```python
# ==================== Scorecardpy评分卡建模 ====================
# 文件名: credit_scoring_scorecardpy.py

import scorecardpy as sc
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
                             f1_score, roc_auc_score, confusion_matrix, roc_curve)
from scipy.stats import ks_2samp
import warnings
warnings.filterwarnings('ignore')

# ==================== 1. 数据加载 ====================
print("="*60)
print("Scorecardpy评分卡建模")
print("="*60)

# 加载数据
app_data = application_record
credit_data = credit_record

print(f"申请数据: {app_data.shape}, 信用记录: {credit_data.shape}")

# ==================== 2. 标签构建 ====================
```

```python
def create_target_label(status):
    """创建目标标签"""
    if status in ['C', 'X', '0']:
        return 1  # Good
    else:
        return 0  # Bad

credit_data['label'] = credit_data['STATUS'].apply(create_target_label)

# 聚合每个客户ID的标签（多数投票）
customer_label = credit_data.groupby('ID')['label'].agg(lambda x: 1 if x.mean() >= 0.5 else 0).reset_index()
customer_label.columns = ['ID', 'label']

print(f"\n标签分布:")
print(f"Good(1): {(customer_label['label']==1).sum():,} ({(customer_label['label']==1).sum()/len(customer_label)*100:.1f}%)")
print(f"Bad(0):  {(customer_label['label']==0).sum():,} ({(customer_label['label']==0).sum()/len(customer_label)*100:.1f}%)")

# ==================== 3. 数据合并与预处理 ====================
# 填充缺失值
app_data['OCCUPATION_TYPE'] = app_data['OCCUPATION_TYPE'].fillna('Unknown')

# 合并数据
data = pd.merge(app_data, customer_label, on='ID', how='inner')
print(f"\n合并后数据: {data.shape}")

# 移除ID列
data = data.drop('ID', axis=1)

# ==================== 4. 数据划分 ====================
print("\n" + "="*60)
print("数据划分")
print("="*60)

# 划分训练测试集
train, test = train_test_split(data, test_size=0.3, random_state=42, stratify=data['label'])

print(f"训练集: {train.shape}")
print(f"测试集: {test.shape}")
print(f"\n训练集标签分布:")
print(train['label'].value_counts())
print(f"\n测试集标签分布:")
print(test['label'].value_counts())

# ==================== 5. Scorecardpy分箱处理 ====================
print("\n" + "="*60)
print("Scorecardpy分箱处理")
print("="*60)

# 自动分箱
bins = sc.woebin(train, y='label',
                 method='freq',        # 等频分箱
```

```
                bin_num_limit=8,      # 最大分箱数
                positive='bad|0',     # 0是bad
                no_cores=4,           # 使用4个CPU核心
                print_step=0)

print(f"分箱完成，特征数: {len(bins)}")

# 计算IV值
iv_table = sc.iv(train, y='label')
print(f"\nIV值统计:")
print(f"IV > 0.3 (强预测力): {len(iv_table[iv_table['info_value'] > 0.3]):,} 个")
print(f"IV 0.1-0.3 (中等预测力): {len(iv_table[(iv_table['info_value'] >= 0.1) & (iv_table['info_value'] <= 0.3)]):,} 个")
print(f"IV 0.02-0.1 (弱预测力): {len(iv_table[(iv_table['info_value'] >= 0.02) & (iv_table['info_value'] < 0.1)]):,} 个")
print(f"IV < 0.02 (无预测力): {len(iv_table[iv_table['info_value'] < 0.02]):,} 个")

print(f"\nIV值最高的10个特征:")
print(iv_table.head(10))

# 基于IV值筛选特征
iv_threshold = 0.02
selected_vars = iv_table[iv_table['info_value'] > iv_threshold].index.tolist()
print(f"\n筛选IV>{iv_threshold}的特征: {len(selected_vars)}个")

# ==================== 6. WOE转换 ====================
print("\n" + "="*60)
print("WOE转换")
print("="*60)

# WOE转换
# 获取列名
selected_columns = [train.columns[i] for i in selected_vars if i < len(train.columns)]
# 确保包含label列
selected_columns_with_label = selected_columns + ['label']

train_woe = sc.woebin_ply(train[selected_columns_with_label], bins, print_step=0)
test_woe = sc.woebin_ply(test[selected_columns_with_label], bins, print_step=0)

# train_woe = sc.woebin_ply(train[selected_vars + ['label']], bins, print_step=0)
# test_woe = sc.woebin_ply(test[selected_vars + ['label']], bins, print_step=0)

print(f"训练集WOE转换后: {train_woe.shape}")
print(f"测试集WOE转换后: {test_woe.shape}")

# 检查数据
if np.any(np.isinf(train_woe.values)) or np.any(np.isinf(test_woe.values)):
    print("处理无穷大值...")
    train_woe = train_woe.replace([np.inf, -np.inf], np.nan)
    test_woe = test_woe.replace([np.inf, -np.inf], np.nan)
    train_woe = train_woe.fillna(0)
    test_woe = test_woe.fillna(0)
```

```python
# ==================== 7. 逻辑回归建模 ====================
print("\n" + "="*60)
print("逻辑回归建模")
print("="*60)

# 准备数据
X_train = train_woe.drop('label', axis=1)
y_train = train_woe['label']
X_test = test_woe.drop('label', axis=1)
y_test = test_woe['label']

print(f"训练集: {X_train.shape}, 测试集: {X_test.shape}")

# 训练逻辑回归
lr = LogisticRegression(C=0.1, max_iter=1000, random_state=42, solver='lbfgs')
lr.fit(X_train, y_train)

# 预测
y_pred = lr.predict(X_test)
y_pred_proba = lr.predict_proba(X_test)[:, 1]

# ==================== 8. 模型评估 ====================
print("\n" + "="*60)
print("模型评估")
print("="*60)

# 计算基础指标
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, zero_division=0)
recall = recall_score(y_test, y_pred, zero_division=0)
f1 = f1_score(y_test, y_pred, zero_division=0)
roc_auc = roc_auc_score(y_test, y_pred_proba)

print(f"准确率: {accuracy:.4f}")
print(f"精确率: {precision:.4f}")
print(f"召回率: {recall:.4f}")
print(f"F1分数: {f1:.4f}")
print(f"ROC-AUC: {roc_auc:.4f}")

# 计算KS值
def calculate_ks(y_true, y_pred_proba):
    good_proba = y_pred_proba[y_true == 1]
    bad_proba = y_pred_proba[y_true == 0]
    if len(good_proba) > 0 and len(bad_proba) > 0:
        ks_stat, _ = ks_2samp(good_proba, bad_proba)
        return ks_stat
    return 0

ks_value = calculate_ks(y_test, y_pred_proba)
print(f"KS值: {ks_value:.4f}")
```

```python
# 混淆矩阵
cm = confusion_matrix(y_test, y_pred)
tn, fp, fn, tp = cm.ravel()
print(f"\n混淆矩阵:")
print(f"TN(正确拒绝坏客户): {tn}")
print(f"FP(错误批准坏客户): {fp}")
print(f"FN(错误拒绝好客户): {fn}")
print(f"TP(正确批准好客户): {tp}")

# 计算误分类成本
bad_loss = 3000  # 坏客户造成的平均损失
good_revenue = 300  # 好客户带来的平均收益
cost = fp * bad_loss + fn * good_revenue
print(f"\n误分类成本: ${cost:,.0f}")

# ==================== 9. 生成评分卡 ====================
print("\n" + "="*60)
print("生成评分卡")
print("="*60)

# 生成评分卡
card = sc.scorecard(bins, lr,
                    # xvariables=selected_vars,
                    xcolumns=selected_columns,  # 使用之前转换的列名
                    points0=600,    # 基准分600
                    odds0=1/30,     # 基准好坏比1:30
                    pdo=20,         # 分数翻倍所需odds变化
                    # basepoints_eq0=False,
                    )

# 计算分数
train_score = sc.scorecard_ply(train, card, print_step=0)
test_score = sc.scorecard_ply(test, card, print_step=0)

# 添加分数到原始数据
train['score'] = train_score['score']
test['score'] = test_score['score']

print(f"训练集分数范围: {train['score'].min():.0f} - {train['score'].max():.0f}")
print(f"训练集平均分: {train['score'].mean():.0f}")
print(f"测试集分数范围: {test['score'].min():.0f} - {test['score'].max():.0f}")
print(f"测试集平均分: {test['score'].mean():.0f}")

# ==================== 10. 模型稳定性评估 ====================
print("\n" + "="*60)
print("模型稳定性评估")
print("="*60)

# 计算PSI
def calculate_psi(train_scores, test_scores, bins=10):
    """计算群体稳定性指标PSI"""
```

```
    # 计算分箱边界
    min_val = min(train_scores.min(), test_scores.min())
    max_val = max(train_scores.max(), test_scores.max())

    # 创建分箱
    train_counts, bin_edges = np.histogram(train_scores, bins=bins, range=(min_val, max_val))
    test_counts, _ = np.histogram(test_scores, bins=bin_edges)

    # 计算比例
    train_ratio = train_counts / len(train_scores)
    test_ratio = test_counts / len(test_scores)

    # 计算PSI
    psi = 0
    for i in range(len(train_ratio)):
        if train_ratio[i] > 0 and test_ratio[i] > 0:
            psi += (test_ratio[i] - train_ratio[i]) * np.log(test_ratio[i] / train_ratio[i])

    return psi

psi = calculate_psi(train['score'], test['score'])
print(f"PSI值: {psi:.4f}")

# 分数分布统计
score_thresholds = [300, 400, 500, 600, 700, 800]
print(f"\n分数分布统计:")
for i in range(len(score_thresholds)-1):
    low, high = score_thresholds[i], score_thresholds[i+1]
    train_count = ((train['score'] >= low) & (train['score'] < high)).sum()
    test_count = ((test['score'] >= low) & (test['score'] < high)).sum()
    train_pct = train_count / len(train) * 100
    test_pct = test_count / len(test) * 100
    print(f"{low}-{high}分: 训练集 {train_count:,} ({train_pct:.1f}%), 测试集 {test_count:,} ({test_pct:.1f}%)")

# ==================== 11. 业务决策分析 ====================
print("\n" + "="*60)
print("业务决策分析")
print("="*60)

# 定义审批阈值
approval_threshold = 600
train_approve_rate = (train['score'] >= approval_threshold).mean() * 100
test_approve_rate = (test['score'] >= approval_threshold).mean() * 100

print(f"审批阈值: {approval_threshold}分")
print(f"训练集审批率: {train_approve_rate:.1f}%")
print(f"测试集审批率: {test_approve_rate:.1f}%")

# 不同阈值下的表现
print(f"\n不同阈值下的表现:")
thresholds = [550, 600, 650, 700]
```

```python
for threshold in thresholds:
    train_bad_rate = train[(train['score'] >= threshold) & (train['label'] == 0)].shape[0] / max(1, train[train['score'] >= threshold].shape[0])
    test_bad_rate = test[(test['score'] >= threshold) & (test['label'] == 0)].shape[0] / max(1, test[test['score'] >= threshold].shape[0])
    print(f"阈值{threshold}分: 训练集坏账率 {train_bad_rate:.2%}, 测试集坏账率 {test_bad_rate:.2%}")

# ==================== 12. 保存结果 ====================
print("\n" + "="*60)
print("保存结果")
print("="*60)

# 保存评分卡
card_df = pd.concat(card)
card_df.to_csv('scorecardpy_scorecard.csv', index=False)
print(f"✓ Scorecardpy评分卡已保存: scorecardpy_scorecard.csv")

# 保存特征分箱结果
bins_df = pd.concat(bins)
bins_df.to_csv('scorecardpy_feature_bins.csv', index=False)
print(f"✓ 特征分箱已保存: scorecardpy_feature_bins.csv")

# 保存预测结果
test_result = test.copy()
test_result['predicted_label'] = y_pred
test_result['predicted_prob'] = y_pred_proba
test_result['score'] = test_score['score']
test_result.to_csv('scorecardpy_test_predictions.csv', index=False)
print(f"✓ 测试集预测结果已保存: scorecardpy_test_predictions.csv")

# 保存IV值
iv_table.to_csv('scorecardpy_feature_iv.csv', index=False)
print(f"✓ 特征IV值已保存: scorecardpy_feature_iv.csv")

# 保存模型系数
coef_df = pd.DataFrame({
    'feature': X_train.columns,
    'coefficient': lr.coef_[0],
    'importance': np.abs(lr.coef_[0])
}).sort_values('importance', ascending=False)
coef_df.to_csv('scorecardpy_model_coefficients.csv', index=False)
print(f"✓ 模型系数已保存: scorecardpy_model_coefficients.csv")

# 保存分数分布
score_distribution = pd.DataFrame({
    'score_range': [f"{i}-{i+100}" for i in range(300, 800, 100)],
    'train_count': [((train['score'] >= i) & (train['score'] < i+100)).sum() for i in range(300, 800, 100)],
    'test_count': [((test['score'] >= i) & (test['score'] < i+100)).sum() for i in range(300, 800, 100)],
    'train_bad_rate': [train[(train['score'] >= i) & (train['score'] < i+100) & (train['label'] == 0)].shape[0] /
                       max(1, train[(train['score'] >= i) & (train['score'] < i+100)].shape[0]) for i in range(300, 800, 100)],
    'test_bad_rate': [test[(test['score'] >= i) & (test['score'] < i+100) & (test['label'] == 0)].shape[0] /
                      max(1, test[(test['score'] >= i) & (test['score'] < i+100)].shape[0]) for i in range(300, 800, 100)]
})
```

```
score_distribution.to_csv('scorecardpy_score_distribution.csv', index=False)
print(f"✓ 分数分布已保存: scorecardpy_score_distribution.csv")

print("\n" + "="*60)
print("Scorecardpy评分卡建模完成")
print("="*60)
print(f"总特征数: {data.shape[1] - 1}")
print(f"评分卡特征数: {len(selected_vars)}")
print(f"模型AUC: {roc_auc:.4f}")
print(f"模型KS: {ks_value:.4f}")
print(f"模型PSI: {psi:.4f}")
print(f"审批阈值: {approval_threshold}分")
print(f"测试集审批率: {test_approve_rate:.1f}%")
```

```
============================================================
Scorecardpy评分卡建模
============================================================
申请数据: (438557, 18), 信用记录: (1048575, 4)

标签分布:
Good(1): 45,847 (99.7%)
Bad(0):  138 (0.3%)

合并后数据: (36457, 19)

============================================================
数据划分
============================================================
训练集: (25519, 18)
测试集: (10938, 18)

训练集标签分布:
label
1    25433
0       86
Name: count, dtype: int64

测试集标签分布:
label
1    10901
0       37
Name: count, dtype: int64

============================================================
Scorecardpy分箱处理
============================================================
[INFO] creating woe binning ...
分箱完成，特征数: 16

IV值统计:
IV > 0.3 (强预测力): 3 个
IV 0.1-0.3 (中等预测力): 5 个
IV 0.02-0.1 (弱预测力): 5 个
IV < 0.02 (无预测力): 4 个

IV值最高的10个特征:
               variable  info_value
9      AMT_INCOME_TOTAL    2.477784
5         DAYS_EMPLOYED    1.391001
13           DAYS_BIRTH    0.635172
1       OCCUPATION_TYPE    0.280208
15      CNT_FAM_MEMBERS    0.193391
8      NAME_INCOME_TYPE    0.175565
11         CNT_CHILDREN    0.173616
6           CODE_GENDER    0.120095
```

```
16  NAME_EDUCATION_TYPE    0.082529
14   NAME_FAMILY_STATUS    0.075639

筛选IV>0.02的特征: 13个

============================================================
WOE转换
============================================================
[INFO] converting into woe values ...
[INFO] converting into woe values ...
训练集WOE转换后: (25519, 14)
测试集WOE转换后: (10938, 14)

============================================================
逻辑回归建模
============================================================
训练集: (25519, 13), 测试集: (10938, 13)

============================================================
模型评估
============================================================
准确率: 0.9966
精确率: 0.9966
召回率: 1.0000
F1分数: 0.9983
ROC-AUC: 0.5828
KS值: 0.2133

混淆矩阵:
TN(正确拒绝坏客户): 0
FP(错误批准坏客户): 37
FN(错误拒绝好客户): 0
TP(正确批准好客户): 10901

误分类成本: $111,000

============================================================
生成评分卡
============================================================
训练集分数范围: 304 - 377
训练集平均分: 336
测试集分数范围: 304 - 372
测试集平均分: 336

============================================================
模型稳定性评估
============================================================
PSI值: 0.0010

分数分布统计:
300-400分: 训练集 25,519 (100.0%), 测试集 10,938 (100.0%)
```

```
400-500分: 训练集 0 (0.0%), 测试集 0 (0.0%)
500-600分: 训练集 0 (0.0%), 测试集 0 (0.0%)
600-700分: 训练集 0 (0.0%), 测试集 0 (0.0%)
700-800分: 训练集 0 (0.0%), 测试集 0 (0.0%)

============================================================
业务决策分析
============================================================
审批阈值: 600分
训练集审批率: 0.0%
测试集审批率: 0.0%

不同阈值下的表现:
阈值550分: 训练集坏账率 0.00%, 测试集坏账率 0.00%
阈值600分: 训练集坏账率 0.00%, 测试集坏账率 0.00%
阈值650分: 训练集坏账率 0.00%, 测试集坏账率 0.00%
阈值700分: 训练集坏账率 0.00%, 测试集坏账率 0.00%

============================================================
保存结果
============================================================
✓ Scorecardpy评分卡已保存: scorecardpy_scorecard.csv
✓ 特征分箱已保存: scorecardpy_feature_bins.csv
✓ 测试集预测结果已保存: scorecardpy_test_predictions.csv
✓ 特征IV值已保存: scorecardpy_feature_iv.csv
✓ 模型系数已保存: scorecardpy_model_coefficients.csv
✓ 分数分布已保存: scorecardpy_score_distribution.csv

============================================================
Scorecardpy评分卡建模完成
============================================================
总特征数: 17
评分卡特征数: 13
模型AUC: 0.5828
模型KS: 0.2133
模型PSI: 0.0010
审批阈值: 600分
测试集审批率: 0.0%
```

In [ ]: