

Credit Risk Analysis

data source : <https://www.kaggle.com/datasets/laotse/credit-risk-dataset>

Feature Name	Description
person_age	Age
person_income	Annual Income
person_home_ownership	Home ownership
person_emp_length	Employment length (in years)
loan_intent	Loan intent
loan_grade	Loan grade
loan_amnt	Loan amount
loan_int_rate	Interest rate
loan_status	Loan status (0 is non default, 1 is default)
loan_percent_income	Percent income
cb_person_default_on_file	Historical default
cb_preson_cred_hist_length	Credit history length

```
In [2]: # Install library
        #!pip install kagglehub
        #!pip install statsmodels
```

Import Library and Setting

```
In [3]: # Import Library
import pandas as pd
import numpy as np
import seaborn as sns
import kagglehub
import matplotlib as mpl
import matplotlib.pyplot as plt
import warnings
import os

from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

# Checking library version
print("---Library version---", end = '\n')
```

```
print('pandas version: ', pd.__version__)
print('numpy version: ', np.__version__)
print('seaborn version: ', sns.__version__)
print('matplotlib version: ', mpl.__version__, end = '\n\n')
```

```
---Library version---
pandas version: 2.2.2
numpy version: 1.26.3
seaborn version: 0.13.2
matplotlib version: 3.9.2
```

```
In [4]: # Setting library
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 100)

mpl.font_manager.fontManager.addfont("fonts\Sarabun-Regular.ttf")
mpl.rc('font', family='Sarabun')
plt.rcParams ['font.family'] = ('Sarabun')

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

```
In [5]: # Print current working directory
print("---Working Directory---", end = '\n')
print('List of Directory:', os.listdir(os.getcwd()))
print('List of Directory (Data):', os.listdir(os.getcwd() + '\\data'))
```

```
---Working Directory---
List of Directory: ['.git', '.ipynb_checkpoints', 'data', 'fonts', 'Risk-Credit-Analysis.ipynb']
List of Directory (Data): ['credit_risk_dataset.csv']
```

Import Data

```
In [6]: source_name = "credit_risk_dataset.csv"

data = pd.read_csv(filepath_or_buffer=f'data/{source_name}')

with pd.option_context('display.max_rows', 10):
    display(data)
```

	person_age	person_income	person_home_ownership	person_emp_length	lo
0	22	59000	RENT	123.0	P
1	21	9600	OWN	5.0	ED
2	25	9600	MORTGAGE	1.0	
3	23	65500	RENT	4.0	
4	24	54400	RENT	8.0	
...	
32576	57	53000	MORTGAGE	1.0	P
32577	54	120000	MORTGAGE	4.0	P
32578	65	76000	RENT	3.0	HOMEIMPRO
32579	56	150000	MORTGAGE	5.0	P
32580	66	42000	RENT	2.0	

32581 rows × 12 columns



```
In [7]: print(f'Records: {data.shape[0]}, Variable: {data.shape[1]}')
```

Records: 32581, Variable: 12

Overall of Data

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32581 entries, 0 to 32580
Data columns (total 12 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   person_age                           32581 non-null  int64
1   person_income                        32581 non-null  int64
2   person_home_ownership                32581 non-null  object
3   person_emp_length                    31686 non-null  float64
4   loan_intent                          32581 non-null  object
5   loan_grade                          32581 non-null  object
6   loan_amnt                           32581 non-null  int64
7   loan_int_rate                       29465 non-null  float64
8   loan_status                         32581 non-null  int64
9   loan_percent_income                 32581 non-null  float64
10  cb_person_default_on_file            32581 non-null  object
11  cb_person_cred_hist_length           32581 non-null  int64
dtypes: float64(3), int64(5), object(4)
memory usage: 3.0+ MB
```

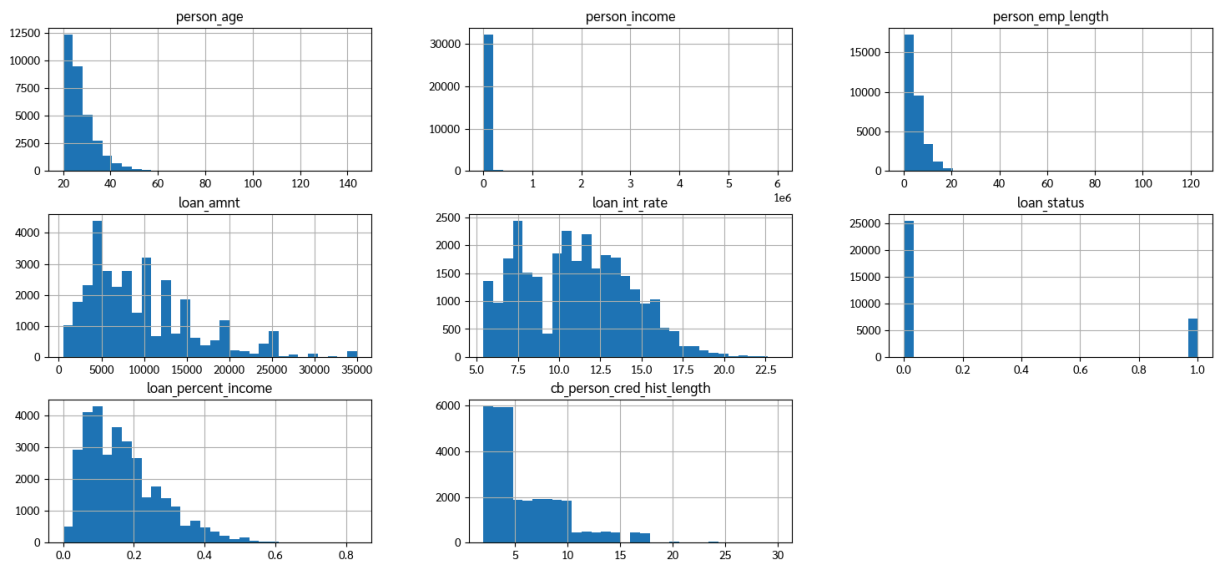
```
In [9]: data.describe()
```

Out[9]:

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status
count	32581.000000	3.258100e+04	31686.000000	32581.000000	29465.000000	3258
mean	27.734600	6.607485e+04	4.789686	9589.371106	11.011695	
std	6.348078	6.198312e+04	4.142630	6322.086646	3.240459	
min	20.000000	4.000000e+03	0.000000	500.000000	5.420000	
25%	23.000000	3.850000e+04	2.000000	5000.000000	7.900000	
50%	26.000000	5.500000e+04	4.000000	8000.000000	10.990000	
75%	30.000000	7.920000e+04	7.000000	12200.000000	13.470000	
max	144.000000	6.000000e+06	123.000000	35000.000000	23.220000	



```
In [10]: data.select_dtypes('number').hist(bins=30, figsize=(18,8))
plt.show()
```



Rename Columns

```
In [11]: df = data.rename(columns=lambda x: x.replace("person_", "").title())
display(df)
```

	Age	Income	Home_Ownership	Emp_Length	Loan_Intent	Loan_Grade	Loar
0	22	59000	RENT	123.0	PERSONAL	D	
1	21	9600	OWN	5.0	EDUCATION	B	
2	25	9600	MORTGAGE	1.0	MEDICAL	C	
3	23	65500	RENT	4.0	MEDICAL	C	
4	24	54400	RENT	8.0	MEDICAL	C	
...
32576	57	53000	MORTGAGE	1.0	PERSONAL	C	
32577	54	120000	MORTGAGE	4.0	PERSONAL	A	
32578	65	76000	RENT	3.0	HOMEIMPROVEMENT	B	
32579	56	150000	MORTGAGE	5.0	PERSONAL	B	
32580	66	42000	RENT	2.0	MEDICAL	B	

32581 rows × 12 columns



Change Columns Type

```
In [12]: df["Emp_Length"] = df["Emp_Length"].astype("Int64")
df["Loan_Int_Rate"] = df["Loan_Int_Rate"].astype("Float64")
```

```
In [13]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32581 entries, 0 to 32580
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  -
 0   Age                   32581 non-null  int64
 1   Income                 32581 non-null  int64
 2   Home_Ownership         32581 non-null  object
 3   Emp_Length             31686 non-null  Int64
 4   Loan_Intent            32581 non-null  object
 5   Loan_Grade             32581 non-null  object
 6   Loan_Amnt              32581 non-null  int64
 7   Loan_Int_Rate          29465 non-null  Float64
 8   Loan_Status            32581 non-null  int64
 9   Loan_Percent_Income    32581 non-null  float64
10   Cb_Default_On_File     32581 non-null  object
11   Cb_Cred_Hist_Length    32581 non-null  int64
dtypes: Float64(1), Int64(1), float64(1), int64(5), object(4)
memory usage: 3.0+ MB
```

Check Missing or NaN value

```
In [14]: for i in df.columns:  
         print(f'{i} : {df[i].isna().sum()}')
```

```
Age : 0  
Income : 0  
Home_Ownership : 0  
Emp_Length : 895  
Loan_Intent : 0  
Loan_Grade : 0  
Loan_Amnt : 0  
Loan_Int_Rate : 3116  
Loan_Status : 0  
Loan_Percent_Income : 0  
Cb_Default_On_File : 0  
Cb_Cred_Hist_Length : 0
```

```
In [15]: print('Object columns with contain NaN values', end='\n\n')  
         for column in df.columns:  
             if df[column].dtype == 'O' and df[column].isna().any():  
                 print(f"NaN in : '{column}'")  
             else:  
                 print("None")
```

Object columns with contain NaN values

```
None  
None  
None  
None  
None  
None  
None  
None  
None  
None  
None  
None  
None
```

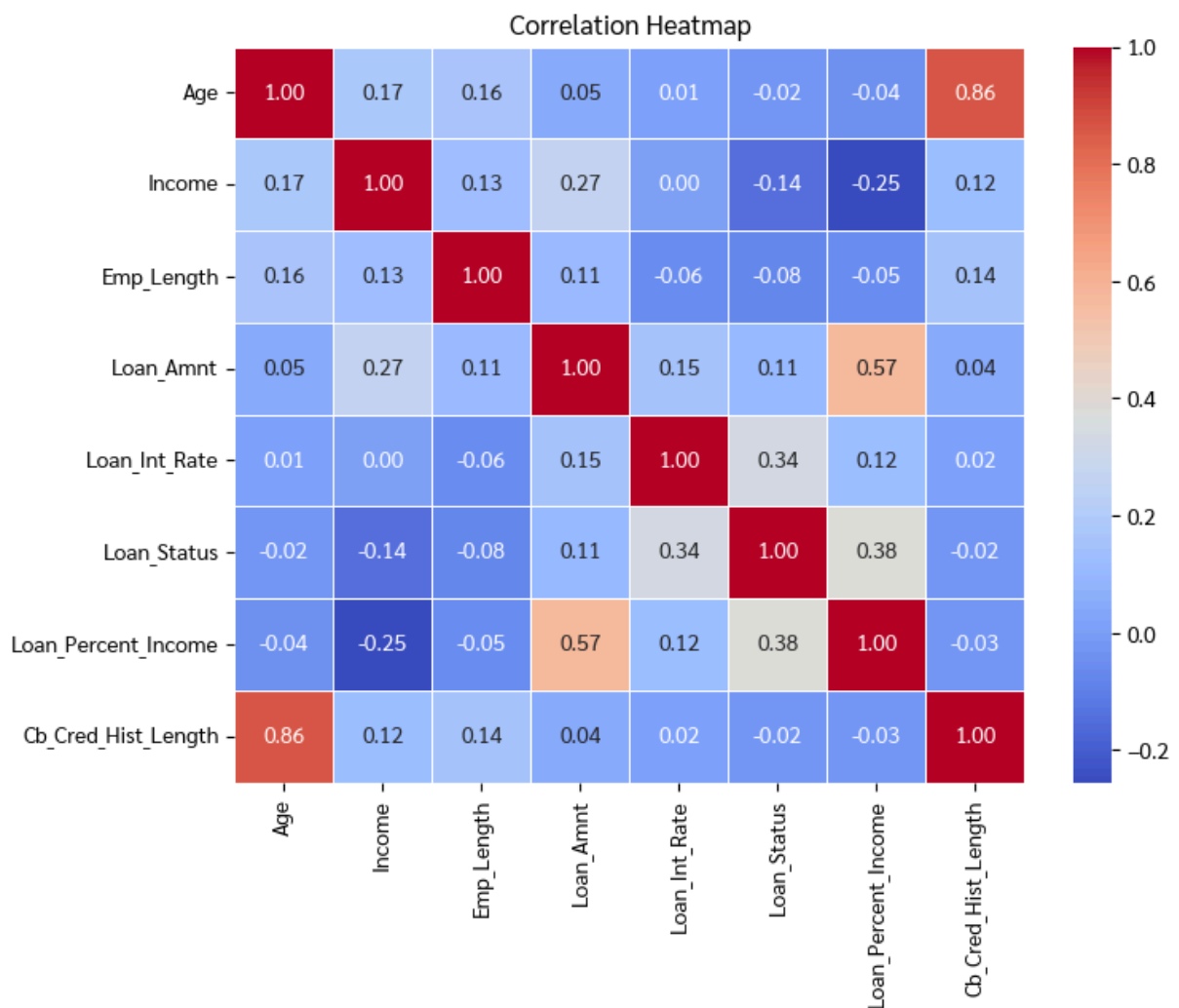
Analysis and Understand insight of the Data

```
In [16]: data.corr(numeric_only=True)
```

Out[16]:

	person_age	person_income	person_emp_length	loan_amnt	loan_int_rate	loan_status	loan_percent_income	cb_person_cred_hist_length
person_age	1.000000	0.173202	0.163106	0.050787	0.012580	-0.021629	-0.042411	0.859133
person_income	0.173202	1.000000	0.134268	0.266820	0.000792	-0.144449	-0.254471	0.117987
person_emp_length	0.163106	0.134268	1.000000	0.113082	-0.056405	-0.082489	-0.054111	0.144699
loan_amnt	0.050787	0.266820	0.113082	1.000000	0.146813	0.105376	0.572612	0.041967
loan_int_rate	0.012580	0.000792	-0.056405	0.146813	1.000000	-0.082489	-0.054111	0.144699
loan_status	-0.021629	-0.144449	-0.082489	0.105376	-0.082489	1.000000	-0.054111	0.144699
loan_percent_income	-0.042411	-0.254471	-0.054111	0.572612	-0.054111	-0.054111	1.000000	0.144699
cb_person_cred_hist_length	0.859133	0.117987	0.144699	0.041967	0.144699	0.144699	0.144699	1.000000

```
In [17]: # Create a heatmap using Seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm', fmt=".2f", lin
plt.title('Correlation Heatmap')
plt.show()
```



Variance Inflation Factor (VIF)

```
In [18]: numeric_df = pd.DataFrame(df.select_dtypes(include=["number"]))
        numeric_df
```

```
Out[18]:
```

	Age	Income	Emp_Length	Loan_Amnt	Loan_Int_Rate	Loan_Status	Loan_Percent_In
0	22	59000	123	35000	16.02	1	
1	21	9600	5	1000	11.14	0	
2	25	9600	1	5500	12.87	1	
3	23	65500	4	35000	15.23	1	
4	24	54400	8	35000	14.27	1	
...
32576	57	53000	1	5800	13.16	0	
32577	54	120000	4	17625	7.49	0	
32578	65	76000	3	35000	10.99	1	
32579	56	150000	5	15000	11.48	0	
32580	66	42000	2	6475	9.99	0	

32581 rows × 8 columns



```
In [19]: # Add constant for intercept
        X = add_constant(df.select_dtypes(include=["number"]).dropna())
```

```
In [20]: X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 28638 entries, 0 to 32580
Data columns (total 9 columns):
#   Column                Non-Null Count  Dtype
---  -
0   const                 28638 non-null  float64
1   Age                   28638 non-null  int64
2   Income                28638 non-null  int64
3   Emp_Length            28638 non-null  int64
4   Loan_Amnt             28638 non-null  int64
5   Loan_Int_Rate         28638 non-null  float64
6   Loan_Status           28638 non-null  int64
7   Loan_Percent_Income   28638 non-null  float64
8   Cb_Cred_Hist_Length    28638 non-null  int64
dtypes: float64(1), int64(1), float64(2), int64(5)
memory usage: 2.2 MB
```

```
In [21]: # Convert all columns to standard float64 to ensure compatibility
X_numeric = X.astype(float)

# Calculating VIF
vif_data = pd.DataFrame()
vif_data["feature"] = X_numeric.columns
vif_data["VIF"] = [variance_inflation_factor(X_numeric.values, i)
                    for i in range(X_numeric.shape[1])]

print(vif_data)
```

	feature	VIF
0	const	56.985541
1	Age	3.941476
2	Income	1.508461
3	Emp_Length	1.064903
4	Loan_Amnt	2.174360
5	Loan_Int_Rate	1.159783
6	Loan_Status	1.346153
7	Loan_Percent_Income	2.367170
8	Cb_Cred_Hist_Length	3.854312

การแปลผลภาพรวม (Interpretation)

โดยทั่วไปเกณฑ์ตัดสินดังนี้:

VIF = 1: ไม่มีความสัมพันธ์กันเลย (ดีมาก)

VIF 1 - 5: มีความสัมพันธ์กันปานกลาง (ยอมรับได้ ไม่เป็นปัญหา)

VIF > 5 หรือ 10: มีความสัมพันธ์กันสูงมาก (อาจเป็นปัญหา) ตัวแปรเหล่านี้อาจซ้ำซ้อนกันเอง ทำให้โมเดลไม่เสถียร

ชุดข้อมูล ไม่มีปัญหา Multicollinearity ที่รุนแรง

ความสัมพันธ์เชิงบวกที่แข็งแกร่งที่สุด (Strongest Positive Correlation)

คู่ที่มีความสัมพันธ์กันชัดเจนที่สุดในตารางคือ:

Age ↔ Cb_Cred_Hist_Length (\$r = 0.86\$)

มีความสัมพันธ์กันสูงมากในทิศทางเดียวกันการตีความ: ยิ่งผู้กู้มีอายุมาก ประวัติเครดิต (Credit History) ก็จะยิ่งยาวนานขึ้น ซึ่งเป็นเรื่องปกติทางธรรมชาติของการสะสมประวัติทางการเงิน

ปัจจัยที่ส่งผลต่อความเสี่ยง (Loan Status Correlations)

หากพิจารณาตัวแปร Loan_Status (สมมติว่าค่าสูง = มีความเสี่ยง/ผิดนัดชำระ) พบความสัมพันธ์ที่น่าสนใจดังนี้:

Loan_Percent_Income (\$r = 0.38\$):

มีความสัมพันธ์เชิงบวกปานกลาง ยิ่งสัดส่วนหนี้ต่อรายได้สูง ความเสี่ยงที่จะเกิดหนี้เสียก็ยิ่งสูงขึ้น (เป็นปัจจัยที่ส่งผลกระทบมากที่สุดต่อ Status ในชุดข้อมูลนี้)

Loan_Int_Rate (\$r = 0.34\$):

อัตราดอกเบี้ยแปรผันตามสถานะสินเชื่อ กลุ่มที่มีความเสี่ยงสูงมักต้องแบกรับดอกเบี้ยที่สูงกว่า หรือดอกเบี้ยที่สูงอาจเป็นตัวเร่งให้เกิดการผิดนัดชำระ

Income (\$r = -0.14\$):

มีความสัมพันธ์เชิงลบเล็กน้อย หมายความว่าผู้ที่มีรายได้สูง มีแนวโน้มจะเป็นหนี้เสียน้อยกว่าผู้ที่มีรายได้ต่ำ แต่ความสัมพันธ์นี้ไม่แรงเท่ากับสัดส่วนหนี้ (Loan Percent Income)

💰 โครงสร้างหนี้และรายได้ (Loan & Income Structure)

ความสัมพันธ์ระหว่างยอดเงินกู้และรายได้ของผู้กู้:

Loan_Amnt ↔ Loan_Percent_Income (\$r = 0.57\$)

มีความสัมพันธ์ค่อนข้างสูง ยิ่งกู้ยืมเงินสูง สัดส่วนภาระหนี้ต่อรายได้ก็จะยิ่งสูงขึ้นตามไปด้วยอย่างชัดเจน

Income ↔ Loan_Percent_Income (\$r = -0.25\$)

มีความสัมพันธ์เชิงลบ (สื่อน้ำเงินเข้ม) แสดงว่าคนที่ รายได้สูง มักจะมีสัดส่วนยอดกู้ต่อรายได้ ต่ำ (ภาระหนี้เบากว่า) ในขณะที่คนรายได้น้อยมักกู้เกินตัวมากกว่า

Weight of Evidence (WoE)

$$\text{WoE}_i = \ln \left(\frac{\% \text{ Non-default}_i}{\% \text{ Default}_i} \right)$$

In [22]: `df.columns`

Out[22]: `Index(['Age', 'Income', 'Home_Ownership', 'Emp_Length', 'Loan_Intent', 'Loan_Grade', 'Loan_Amnt', 'Loan_Int_Rate', 'Loan_Status', 'Loan_Percent_Income', 'Cb_Default_On_File', 'Cb_Cred_Hist_Length'], dtype='object')`

```
In [23]: def calculate_woe_iv(dataset, feature, target):
    lst = []
    # จัดกลุ่มข้อมูลตาม Feature และนับจำนวน Good (0) และ Bad (1)
    for i in range(dataset[feature].nunique()):
        val = list(dataset[feature].unique())[i]
        lst.append({
            'Value': val,
            'All': dataset[dataset[feature] == val].count()[feature],
            'Good': dataset[(dataset[feature] == val) & (dataset[target] == 0)].count(),
            'Bad': dataset[(dataset[feature] == val) & (dataset[target] == 1)].count()
        })
```

```

    })

dffb = pd.DataFrame(lst)
dffb['Dist_Good'] = dffb['Good'] / dffb['Good'].sum()
dffb['Dist_Bad'] = dffb['Bad'] / dffb['Bad'].sum()

# คำนวณ WoE
dffb['WoE'] = np.log(dffb['Dist_Good'] / dffb['Dist_Bad'])
dffb.replace({'WoE': {np.inf: 0, -np.inf: 0}}, inplace=True)

# คำนวณ IV
dffb['IV'] = (dffb['Dist_Good'] - dffb['Dist_Bad']) * dffb['WoE']
iv = dffb['IV'].sum()

return dffb, iv

```

```

In [24]: def plot_woe(woe_df, feature_name):
plt.figure(figsize=(10, 6))

# พล็อตกราฟเส้นเพื่อดูแนวโน้ม (Trend)
sns.pointplot(x='Value', y='WoE', data=woe_df, color='#1f77b4')

# ตกแต่งกราฟ
plt.title(f'Weight of Evidence (WoE) for {feature_name}', fontsize=14)
plt.axhline(y=0, color='red', linestyle='--', alpha=0.5) # เส้นแบ่งความเสี่ยงปกติ
plt.xticks(rotation=45)
plt.grid(axis='y', alpha=0.3)
plt.ylabel('WoE Value')
plt.xlabel(f'Bins of {feature_name}')

plt.tight_layout()
plt.show()

```

```

In [25]: def calculate_ks(woe_df):
# ตรวจสอบให้แน่ใจว่าเรียงลำดับตามความเสี่ยงแล้ว
df_ks = woe_df.sort_values(by='WoE').reset_index(drop=True)

# คำนวณ % สะสม
df_ks['Cumulative_Good'] = df_ks['Dist_Good'].cumsum()
df_ks['Cumulative_Bad'] = df_ks['Dist_Bad'].cumsum()

# คำนวณส่วนต่าง
df_ks['KS_Diff'] = np.abs(df_ks['Cumulative_Good'] - df_ks['Cumulative_Bad'])

ks_value = df_ks['KS_Diff'].max()
return ks_value, df_ks

```

Calculate WOE

ตัวแปร 'person_home_ownership'

```

In [26]: # 'person_home_ownership'
woe_df, iv_value = calculate_woe_iv(df, 'Home_Ownership', 'Loan_Status')
print(f"Information Value: {iv_value}")

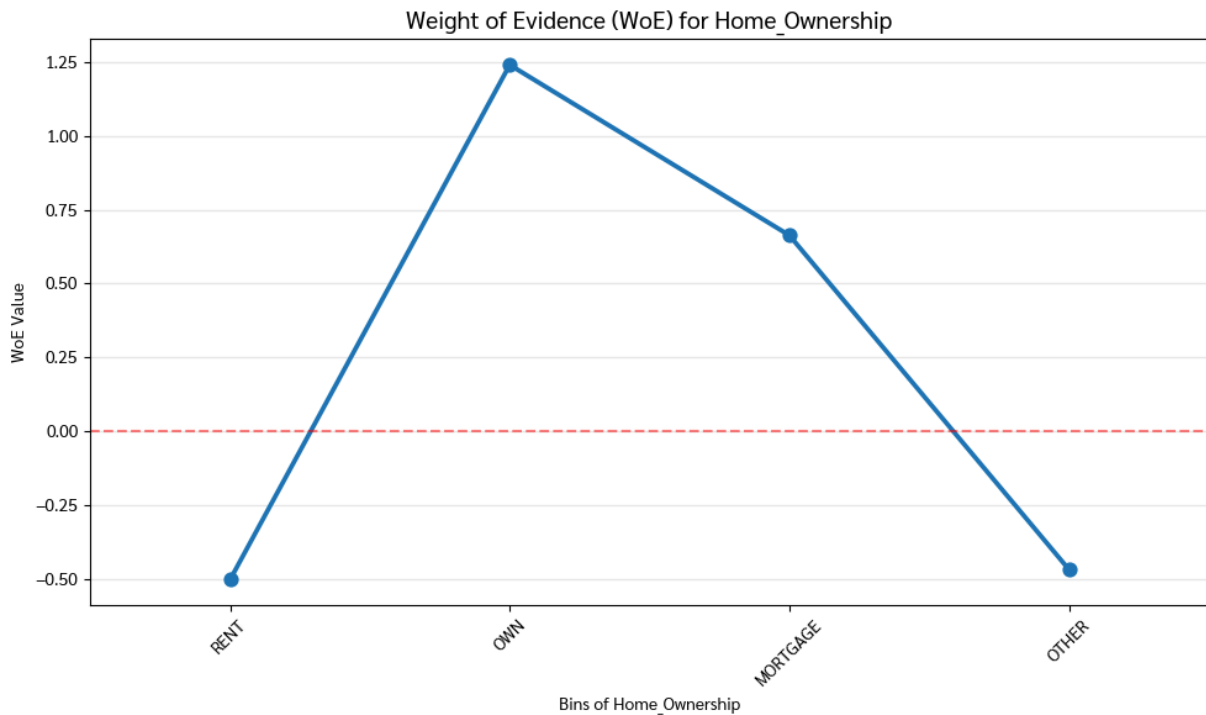
```

```
print(woe_df)

plot_woe(woe_df, 'Home_Ownership')
```

Information Value: 0.37699817947172454

	Value	All	Good	Bad	Dist_Good	Dist_Bad	WoE	IV
0	RENT	16446	11254	5192	0.441801	0.730445	-0.502794	0.145128
1	OWN	2584	2391	193	0.093864	0.027153	1.240379	0.082748
2	MORTGAGE	13444	11754	1690	0.461430	0.237760	0.663067	0.148308
3	OTHER	107	74	33	0.002905	0.004643	-0.468841	0.000815



```
In [27]: ks_score, ks_table = calculate_ks(woe_df)
print(f"KS Statistic: {ks_score:.4f}")
```

KS Statistic: 0.2904

ตัวแปร 'income_bin'

```
In [28]: # แบ่งรายได้ (Income) เป็น 5 กลุ่ม กลุ่มละเท่าๆ กัน (Quantile-based)
df['income_bin'] = pd.qcut(df['Income'], q=5, precision=0)
```

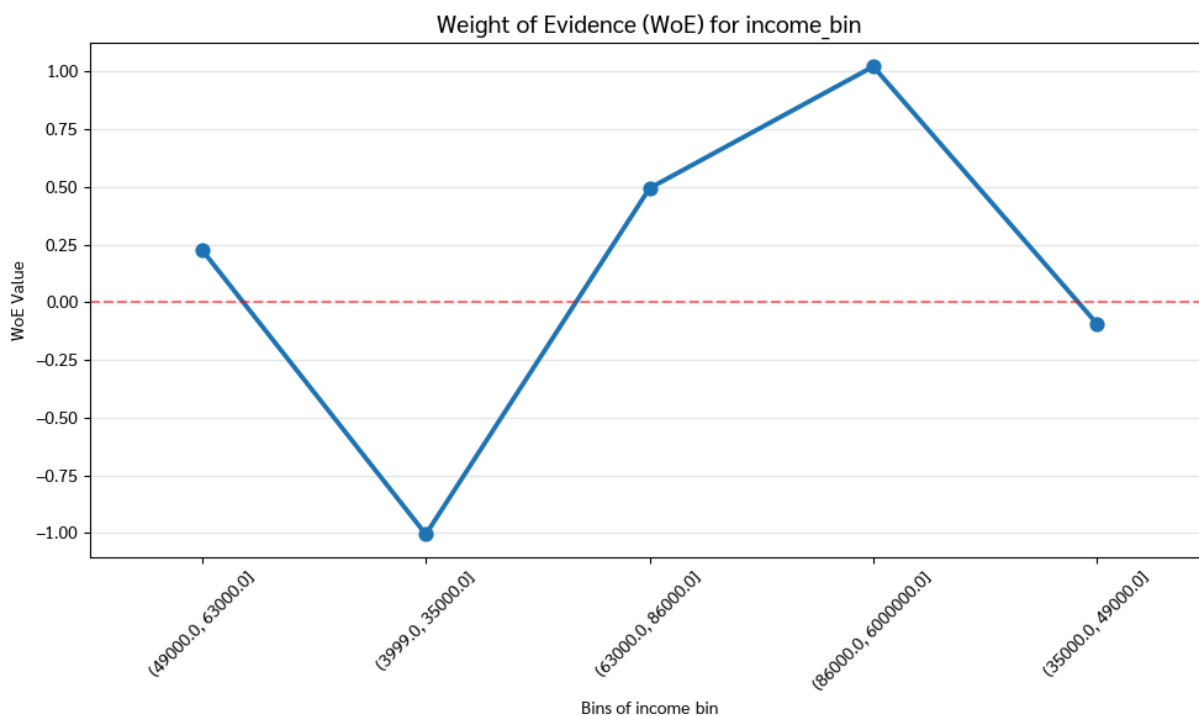
```
In [29]: # 'income_bin'
woe_df, iv_value = calculate_woe_iv(df, 'income_bin', 'Loan_Status')
print(f"Information Value: {iv_value}")
print(woe_df)

plot_woe(woe_df, 'income_bin')
```

Information Value: 0.4617250888756999

	Value	All	Good	Bad	Dist_Good	Dist_Bad	WoE \
0	(49000.0, 63000.0]	6397	5233	1164	0.205433	0.163759	0.226724
1	(3999.0, 35000.0]	6630	3762	2868	0.147686	0.403489	-1.005062
2	(63000.0, 86000.0]	6515	5567	948	0.218545	0.133371	0.493859
3	(86000.0, 600000.0]	6493	5900	593	0.231618	0.083427	1.021115
4	(35000.0, 49000.0]	6546	5011	1535	0.196718	0.215954	-0.093293

	IV
0	0.009449
1	0.257098
2	0.042064
3	0.151320
4	0.001795



```
In [30]: ks_score, ks_table = calculate_ks(woe_df)
print(f"KS Statistic: {ks_score:.4f}")
```

KS Statistic: 0.2750

ตัวแปร 'Loan_Int_Rate'

```
In [31]: df_woe = df.copy()

# แบ่ง Loan_Int_Rate เป็น 5 กลุ่ม กลุ่มละเท่าๆ กัน (Quantile-based)
df_woe['Loan_Int_Rate_bin'] = pd.qcut(df_woe['Loan_Int_Rate'], q=5, precision=0)
```

```
In [32]: # Add 'Missing' category and fill
df_woe['Loan_Int_Rate_bin'] = df_woe['Loan_Int_Rate_bin'].cat.add_categories(['Missing'])
df_woe['Loan_Int_Rate_bin'] = df_woe['Loan_Int_Rate_bin'].fillna('Missing')
```

```
In [33]: print("\nDistribution of Loan_Int_Rate_bin:")
print(df_woe['Loan_Int_Rate_bin'].value_counts())
```

Distribution of Loan_Int_Rate_bin:

```
Loan_Int_Rate_bin
(4.0, 8.0]      6016
(8.0, 10.0]     5988
(14.0, 23.0]    5868
(12.0, 14.0]    5814
(10.0, 12.0]    5779
Missing         3116
Name: count, dtype: int64
```

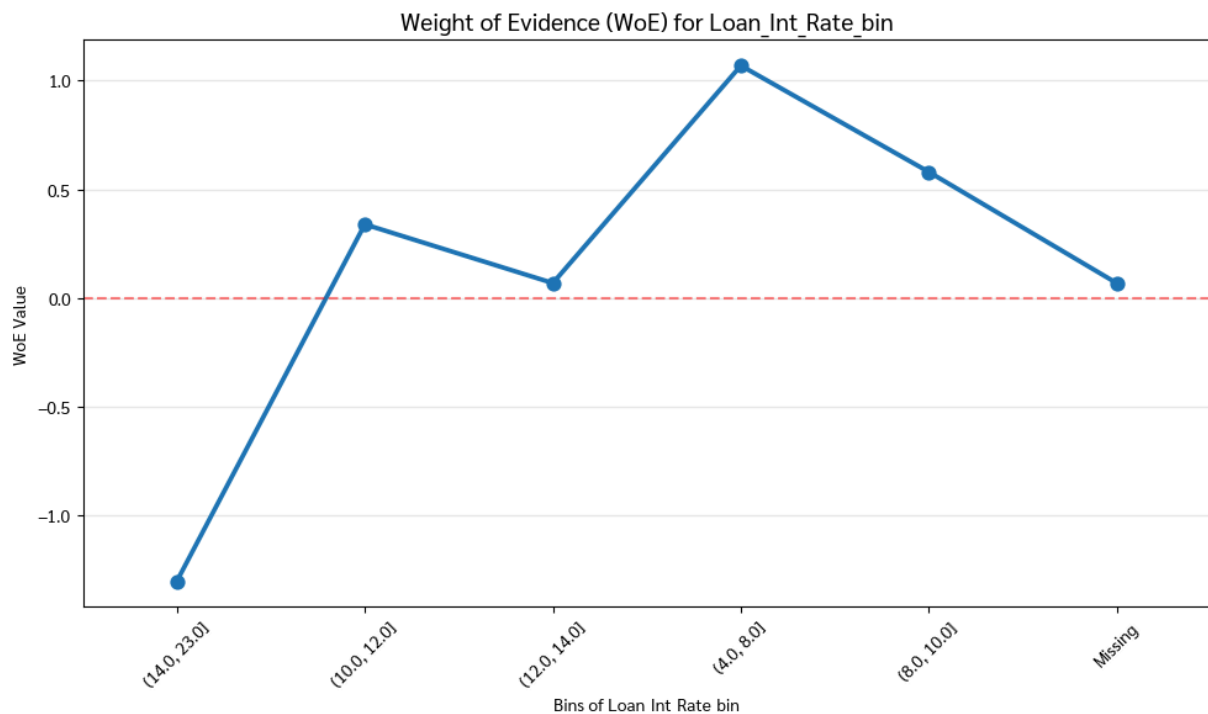
```
In [34]: # Calculate WOE for Emp_Length with the new Missing category
woe_emp, iv_emp = calculate_woe_iv(df_woe, 'Loan_Int_Rate_bin', 'Loan_Status')

print(f"\nInformation Value (Emp_Length): {iv_emp:.4f}")
# Sort by WoE to see where 'Missing' falls in terms of risk
print(woe_emp.sort_values(by='WoE'))

# Plotting
plot_woe(woe_emp, 'Loan_Int_Rate_bin')
```

Information Value (Emp_Length): 0.6186

	Value	All	Good	Bad	Dist_Good	Dist_Bad	WoE	IV
0	(14.0, 23.0]	5868	2898	2970	0.113768	0.417839	-1.300939	0.395579
5	Missing	3116	2472	644	0.097044	0.090602	0.068686	0.000442
2	(12.0, 14.0]	5814	4613	1201	0.181094	0.168965	0.069326	0.000841
1	(10.0, 12.0]	5779	4821	958	0.189259	0.134778	0.339491	0.018496
4	(8.0, 10.0]	5988	5179	809	0.203313	0.113815	0.580170	0.051924
3	(4.0, 8.0]	6016	5490	526	0.215522	0.074001	1.068984	0.151284



```
In [35]: ks_score, ks_table = calculate_ks(woe_emp)
print(f"KS Statistic: {ks_score:.4f}")
```

KS Statistic: 0.3041

IV > 0.5 ถือว่า "สูง/ผิดปกติ (อาจเกิดการโอเวอร์ฟิตติ้งหรือการรั่วไหลของข้อมูล)" ในการประเมินความเสี่ยงด้านเครดิต อัตราดอกเบี้ยมักจะถูกกำหนดหลังจากประเมินความเสี่ยงแล้ว

ตัวแปร 'Age'

```
In [36]: # การแบ่งช่วงอายุ (Age) ตามเกณฑ์ที่กำหนดเอง
bins = [0, 25, 35, 45, 60, 100]
labels = ['<25', '25-35', '35-45', '45-60', '>60']

df['age_bin'] = pd.cut(df['Age'], bins=bins, labels=labels)
```

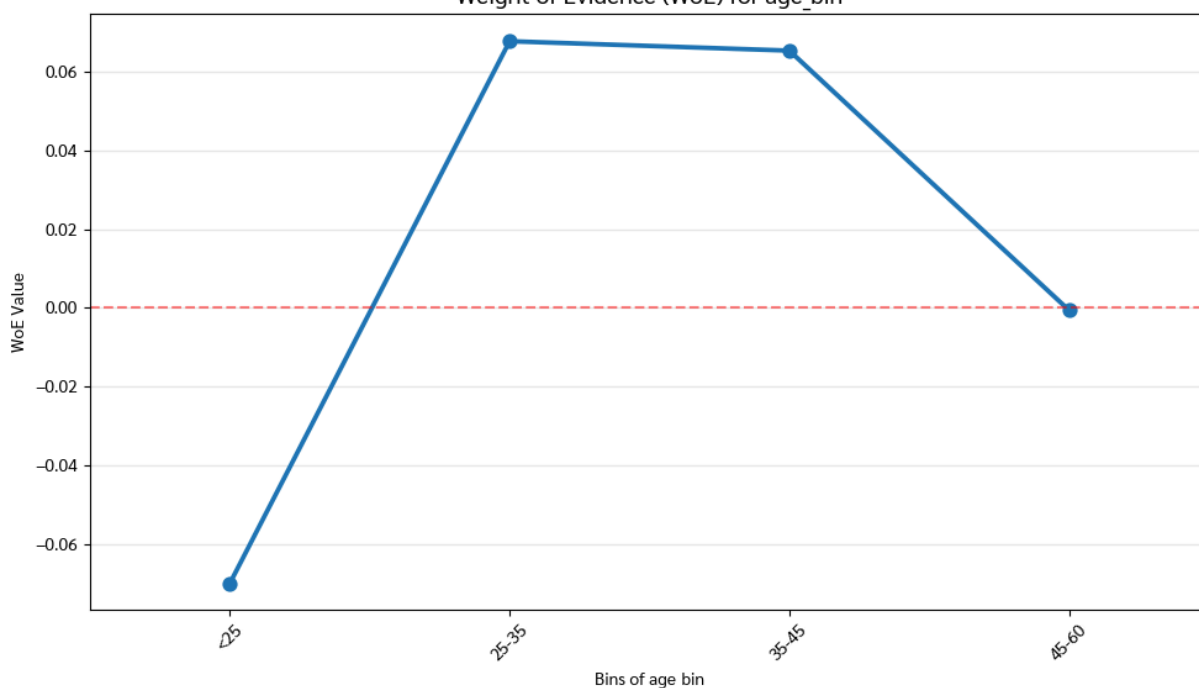
```
In [37]: woe_df, iv_value = calculate_woe_iv(df, 'age_bin', 'Loan_Status')
print(f"Information Value: {iv_value}")
print(woe_df)

plot_woe(woe_df, 'age_bin')
```

Information Value: 0.004616946069285549

	Value	All	Good	Bad	Dist_Good	Dist_Bad	WoE	IV
0	<25	15352	11817	3535	0.464870	0.498519	-0.069884	2.351532e-03
1	25-35	13763	10917	2846	0.429465	0.401354	0.067697	1.903036e-03
2	NaN	0	0	0	0.000000	0.000000	NaN	NaN
3	35-45	2814	2231	583	0.087766	0.082217	0.065308	3.623715e-04
4	45-60	582	455	127	0.017899	0.017910	-0.000600	6.436204e-09

Weight of Evidence (WoE) for age_bin



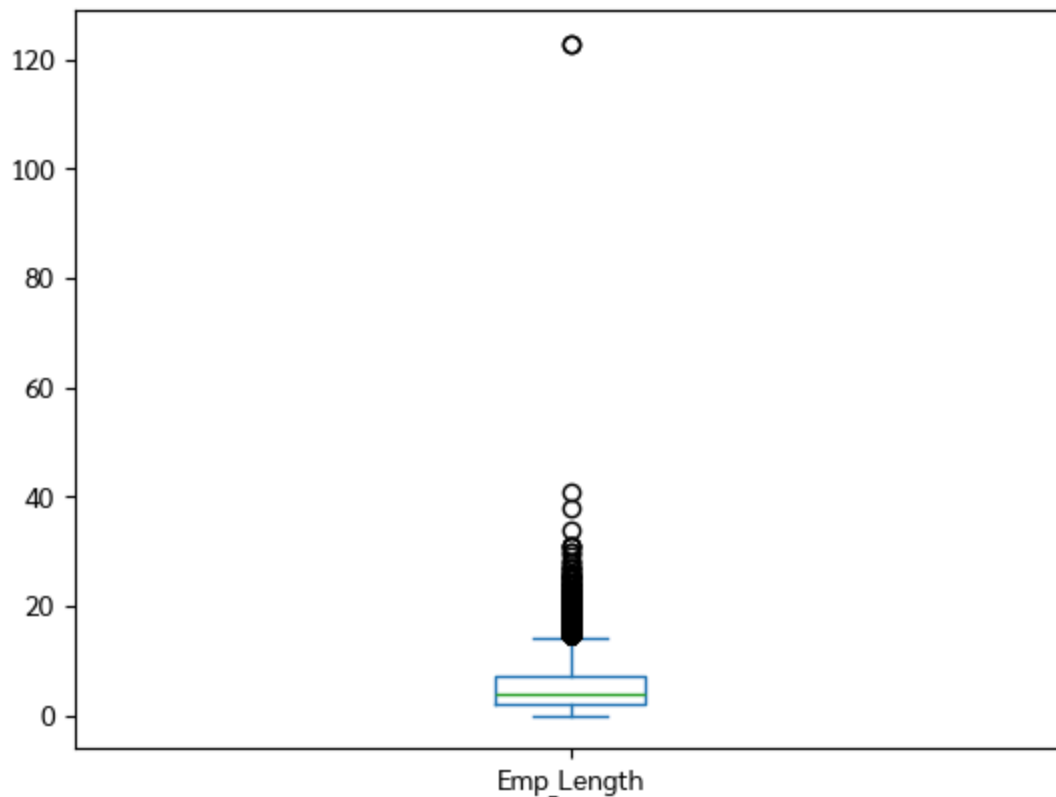
```
In [38]: ks_score, ks_table = calculate_ks(woe_df)
print(f"KS Statistic: {ks_score:.4f}")
```

KS Statistic: 0.0337

ตัวแปร 'Emp_Length'

```
In [39]: df['Emp_Length'].dropna().plot.box()
```

```
Out[39]: <Axes: >
```



```
In [40]: df['Emp_Length'].max()
```

```
Out[40]: 123
```

```
In [41]: # Make a copy to avoid SettingWithCopy warnings
df_woe = df.copy()

bins_emp = [0, 2, 4, 6, 8, 10, 40, float('inf')]
labels_emp = ['<2', '2-4', '4-6', '6-8', '8-10', '10-40', '>40']

# Create the bins
df_woe['Emp_Length_bin'] = pd.cut(df_woe['Emp_Length'], bins=bins_emp, labels=labels_emp)
```

```
In [42]: # Add 'Missing' to the category list and then fill NaNs
df_woe['Emp_Length_bin'] = df_woe['Emp_Length_bin'].cat.add_categories(['Missing'])
df_woe['Emp_Length_bin'] = df_woe['Emp_Length_bin'].fillna('Missing')
```

```
In [43]: # Check the distribution (Missing should now appear in the List)
print("Distribution of Emp_Length_bin:")
print(df_woe['Emp_Length_bin'].value_counts())
```

Distribution of Emp_Length_bin:

```
Emp_Length_bin
2-4          7305
<2           7020
4-6          5820
6-8          4862
10-40        3622
8-10         3054
Missing       895
>40           3
```

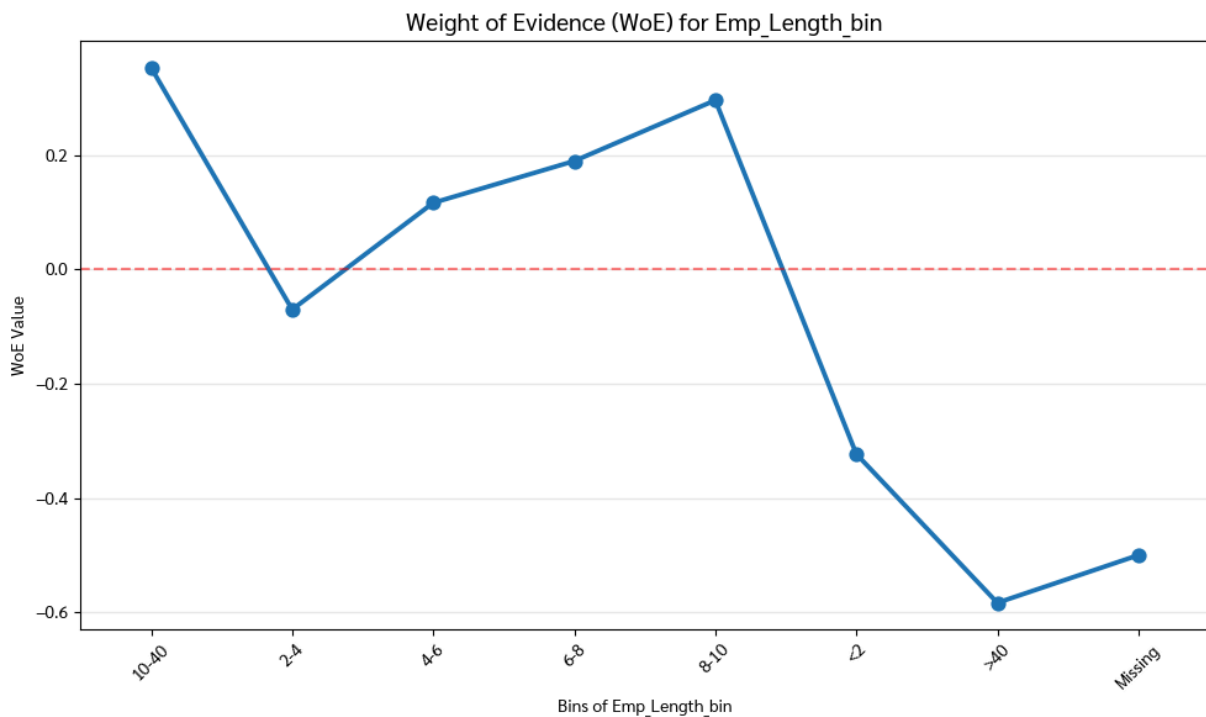
Name: count, dtype: int64

```
In [44]: woe_df, iv_value = calculate_woe_iv(df_woe, 'Emp_Length_bin', 'Loan_Status')
woe_df = woe_df.sort_values('Value')
print(f"Information Value: {iv_value}")
print(woe_df)

plot_woe(woe_df, 'Emp_Length_bin')
```

Information Value: 0.06083576424417617

	Value	All	Good	Bad	Dist_Good	Dist_Bad	WoE	IV
6	10-40	3622	3028	594	0.118871	0.083568	0.352380	0.012440
4	2-4	7305	5622	1683	0.220704	0.236775	-0.070289	0.001130
1	4-6	5820	4662	1158	0.183017	0.162915	0.116352	0.002339
5	6-8	4862	3950	912	0.155066	0.128306	0.189433	0.005069
3	8-10	3054	2529	525	0.099282	0.073860	0.295783	0.007519
2	<2	7020	5067	1953	0.198916	0.274761	-0.323016	0.024499
0	>40	3	2	1	0.000079	0.000141	-0.583251	0.000036
7	Missing	895	613	282	0.024065	0.039674	-0.499940	0.007804

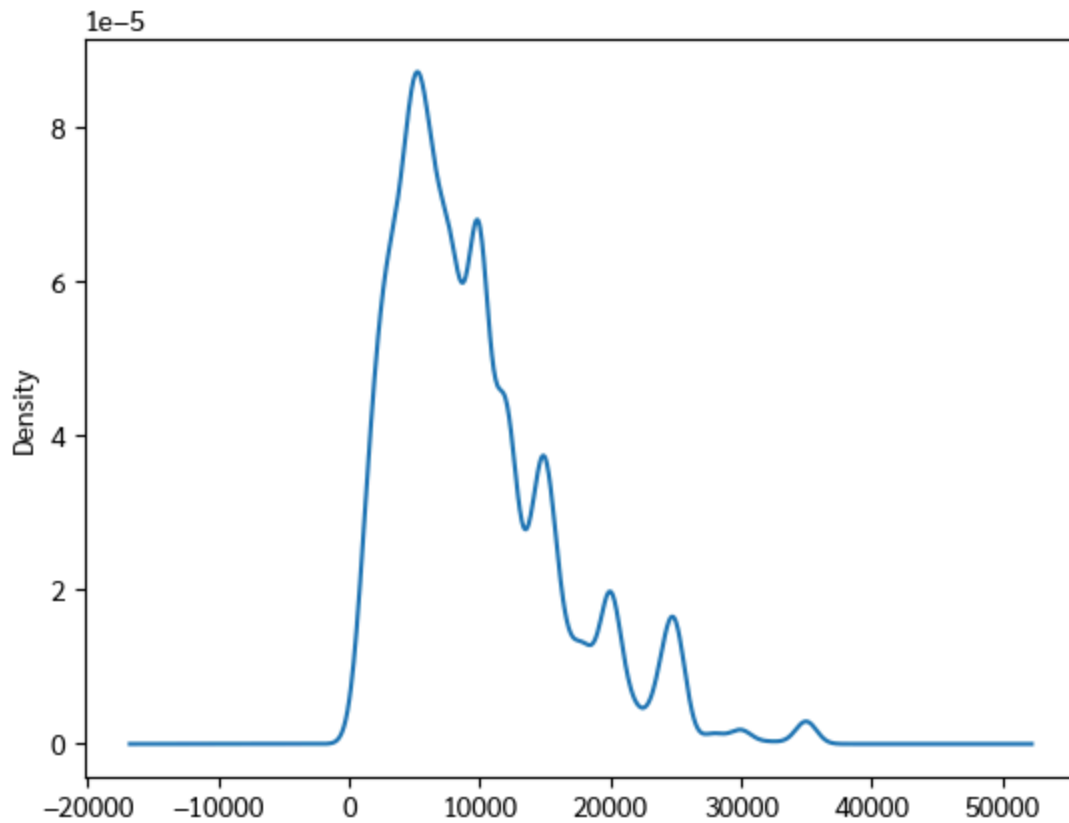


```
In [45]: ks_score, ks_table = calculate_ks(woe_df)
print(f"KS Statistic: {ks_score:.4f}")
```

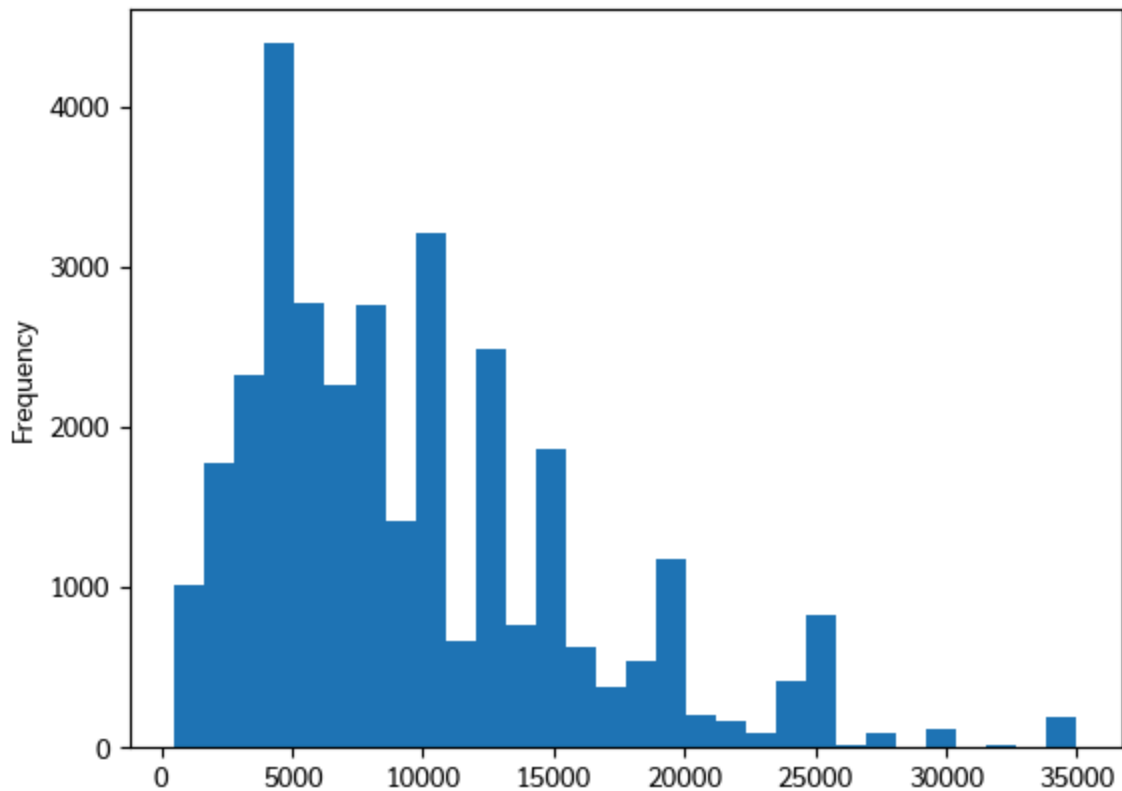
KS Statistic: 0.1076

ตัวแปร 'loan_amnt'

```
In [46]: # single column (density)
df['Loan_Amnt'].plot(kind='kde')
plt.show()
```



```
In [47]: df['Loan_Amnt'].plot(kind='hist', bins=30)
plt.show()
```



```
In [48]: # Make a copy to avoid SettingWithCopy warnings
df_woe = df.copy()

# Make a copy to avoid SettingWithCopy warnings
df_woe['Loan_Amnt_bin'] = pd.qcut(df_woe['Loan_Amnt'], q=5, precision=0)
```

```
In [49]: print("\nDistribution of Loan_Amnt_bin:")
print(df_woe['Loan_Amnt_bin'].value_counts())
```

Distribution of Loan_Amnt_bin:

```
Loan_Amnt_bin
(6750.0, 10000.0]    8493
(499.0, 4400.0]     6543
(4400.0, 6750.0]    6493
(14500.0, 35000.0]  6493
(10000.0, 14500.0]  4559
Name: count, dtype: int64
```

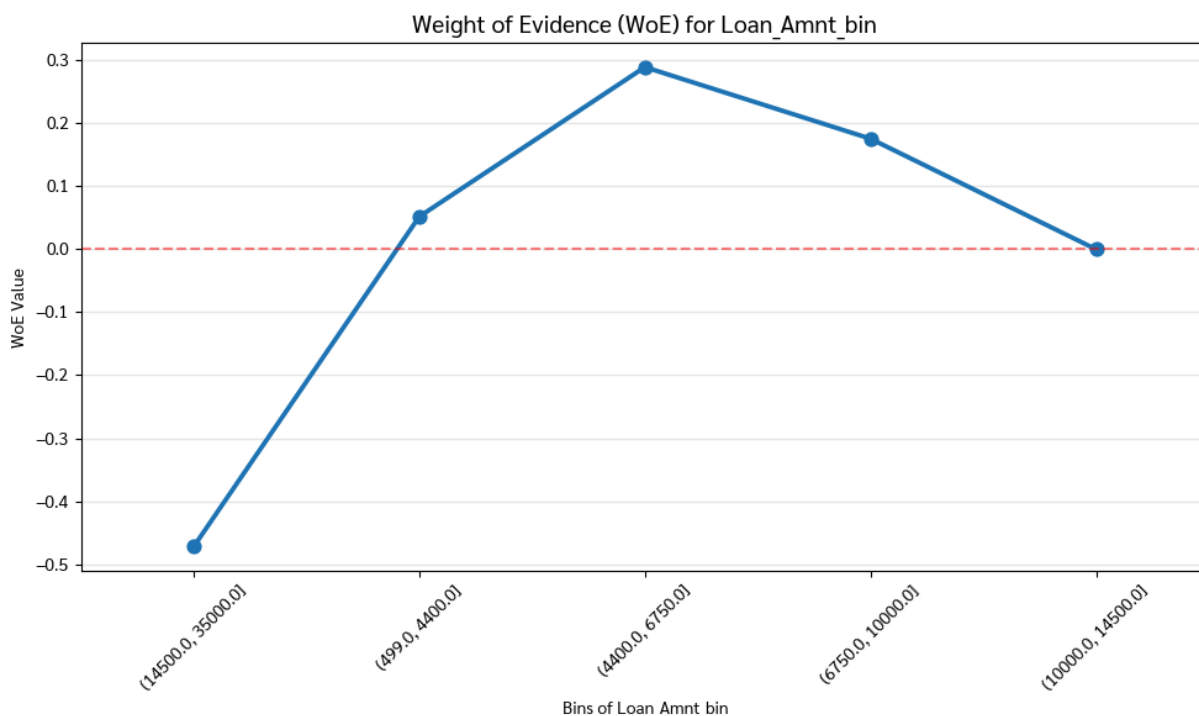
```
In [50]: # 'Loan_Amnt_bin'
woe_df, iv_value = calculate_woe_iv(df_woe, 'Loan_Amnt_bin', 'Loan_Status')
print(f"Information Value: {iv_value}")
print(woe_df)

plot_woe(woe_df, 'Loan_Amnt_bin')
```

Information Value: 0.07349448129649444

	Value	All	Good	Bad	Dist_Good	Dist_Bad	WoE \
0	(14500.0, 35000.0]	6493	4486	2007	0.176108	0.282358	-0.472078
1	(499.0, 4400.0]	6543	5172	1371	0.203039	0.192881	0.051321
2	(4400.0, 6750.0]	6493	5370	1123	0.210811	0.157991	0.288426
3	(6750.0, 10000.0]	8493	6881	1612	0.270129	0.226787	0.174890
4	(10000.0, 14500.0]	4559	3564	995	0.139913	0.139983	-0.000502

	IV
0	5.015820e-02
1	5.212793e-04
2	1.523480e-02
3	7.580167e-03
4	3.528234e-08



ตัวแปร 'Loan_Percent_Income_bin'

```
In [54]: df['Loan_Percent_Income_bin'] = pd.qcut(df['Loan_Percent_Income'], q=5, precision=2)
         woe_lpi, iv_lpi = calculate_woe_iv(df, 'Loan_Percent_Income_bin', 'Loan_Status')
```

```
In [56]: # Check the distribution (Missing should now appear in the List)
         print("Distribution of Loan_Percent_Income_bin:")
         print(df['Loan_Percent_Income_bin'].value_counts())
```

Distribution of Loan_Percent_Income_bin:

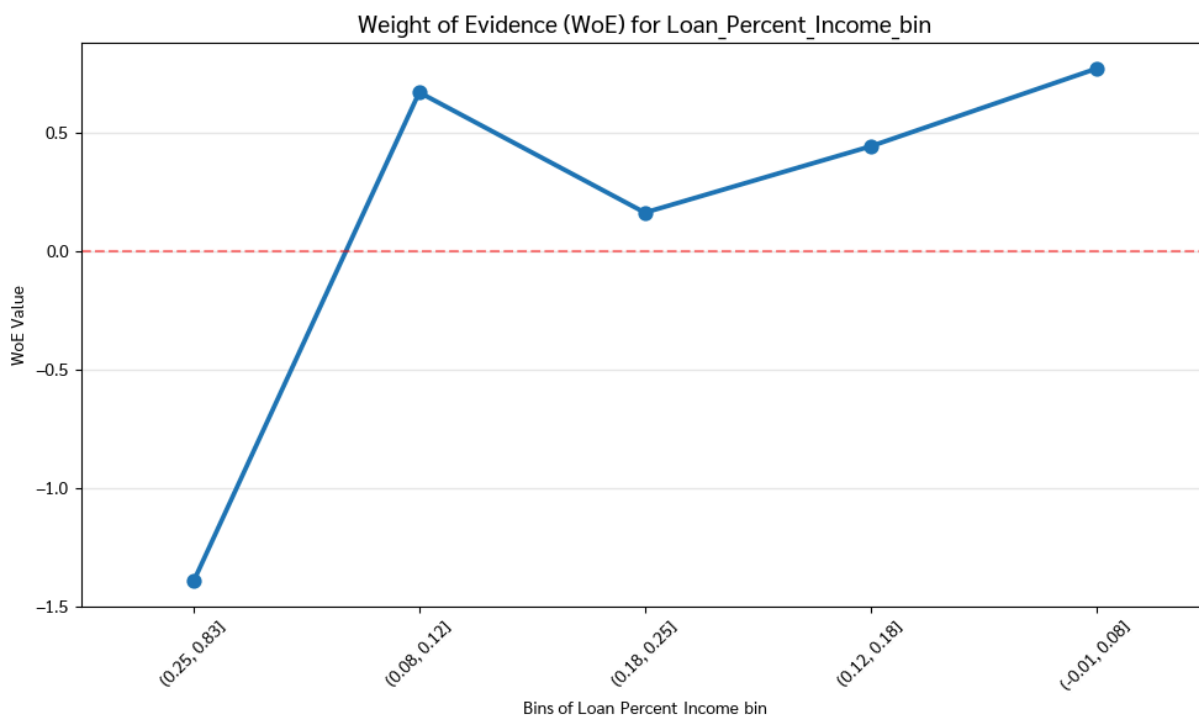
```
Loan_Percent_Income_bin
(-0.01, 0.08]      7572
(0.12, 0.18]      7343
(0.25, 0.83]      6280
(0.18, 0.25]      5800
(0.08, 0.12]      5586
Name: count, dtype: int64
```

```
In [57]: # 'Loan_Amnt_bin'
woe_df, iv_value = calculate_woe_iv(df, 'Loan_Percent_Income_bin', 'Loan_Status')
print(f"Information Value: {iv_value}")
print(woe_df)

plot_woe(woe_df, 'Loan_Percent_Income_bin')
```

Information Value: 0.7066241741715152

	Value	All	Good	Bad	Dist_Good	Dist_Bad	WoE	IV
0	(0.25, 0.83]	6280	2956	3324	0.116044	0.467642	-1.393730	0.490032
1	(0.08, 0.12]	5586	4890	696	0.191968	0.097918	0.673200	0.063315
2	(0.18, 0.25]	5800	4690	1110	0.184117	0.156162	0.164674	0.004603
3	(0.12, 0.18]	7343	6229	1114	0.244533	0.156725	0.444861	0.039063
4	(-0.01, 0.08]	7572	6708	864	0.263338	0.121553	0.773085	0.109611



```
In [58]: # 2. Loan_Intent (Categorical)
woe_intent, iv_intent = calculate_woe_iv(df, 'Loan_Intent', 'Loan_Status')

# 3. Cb_Default_On_File (Categorical)
woe_cb, iv_cb = calculate_woe_iv(df, 'Cb_Default_On_File', 'Loan_Status')
```

การแปลความหมายค่า KS

ค่า KS จะมีค่าอยู่ระหว่าง 0 ถึง 1 (หรือ 0% ถึง 100%):

- < 20%: ความสามารถในการแยกแยะต่ำมาก (โมเดลใช้ไม่ได้)
- 20% - 40%: ความสามารถในการแยกแยะปานกลาง (ยอมรับได้สำหรับงานทั่วไป)
- 40% - 50%: ดีมาก (Good) เป็นช่วงที่เหมาะสมสำหรับ Credit Scoring
- 50% - 70%: ยอดเยี่ยม (Excellent)

70%>: สูงผิดปกติ (อาจเกิด Overfitting หรือตัวแปรนั้นมีเจลยอยู่ในตัวแล้ว เช่น เอาคอลัมน์ "ยอดค้างชำระปัจจุบัน" ไปทำนาย "การเบี้ยวหนี้")

เกณฑ์การแปลความหมาย IV:

- <0.02: ไม่มีอำนาจในการพยากรณ์
- 0.02 - 0.1: อำนาจการพยากรณ์ต่ำ
- 0.1 - 0.3: อำนาจการพยากรณ์ปานกลาง
- 0.3 - 0.5: อำนาจการพยากรณ์สูง
- 0.5>: สูงผิดปกติ (อาจเกิด Overfitting หรือ Data Leakage)

Logistic Regression Model

```
In [60]: # --- 1. Consolidate Binning into Main DataFrame (df) ---

# Income Binning
df['income_bin'] = pd.qcut(df['Income'], q=5, precision=0)

# Emp_Length Binning (Your logic from the screenshot, applied to df)
bins_emp = [0, 2, 4, 6, 8, 10, 40, float('inf')]
labels_emp = ['<2', '2-4', '4-6', '6-8', '8-10', '10-40', '>40']
df['Emp_Length_bin'] = pd.cut(df['Emp_Length'], bins=bins_emp, labels=labels_emp, right=True)
df['Emp_Length_bin'] = df['Emp_Length_bin'].cat.add_categories(['Missing']).fillna('Missing')

# Loan Amount Binning
df['Loan_Amnt_bin'] = pd.qcut(df['Loan_Amnt'], q=5, precision=0)

# Loan Percent Income Binning (Adding this as it's a strong predictor)
df['Loan_Percent_Income_bin'] = pd.qcut(df['Loan_Percent_Income'], q=5, precision=2)

print("Binning complete. Columns in df:", df.columns)
```

```
Binning complete. Columns in df: Index(['Age', 'Income', 'Home_Ownership', 'Emp_Length', 'Loan_Intent',
      'Loan_Grade', 'Loan_Amnt', 'Loan_Int_Rate', 'Loan_Status',
      'Loan_Percent_Income', 'Cb_Default_On_File', 'Cb_Cred_Hist_Length',
      'income_bin', 'age_bin', 'Loan_Percent_Income_bin', 'Emp_Length_bin',
      'Loan_Amnt_bin'],
      dtype='object')
```

```
In [61]: # --- 2. Calculate WOE for each feature and save to unique variables ---

# Home Ownership
woe_home, iv_home = calculate_woe_iv(df, 'Home_Ownership', 'Loan_Status')

# Income
woe_income, iv_income = calculate_woe_iv(df, 'income_bin', 'Loan_Status')
```

```

# Emp Length
woe_emp, iv_emp = calculate_woe_iv(df, 'Emp_Length_bin', 'Loan_Status')

# Loan Amount
woe_amnt, iv_amnt = calculate_woe_iv(df, 'Loan_Amnt_bin', 'Loan_Status')

# Loan Percent Income
woe_lpi, iv_lpi = calculate_woe_iv(df, 'Loan_Percent_Income_bin', 'Loan_Status')

print("WOE Tables ready.")

```

WOE Tables ready.

```

In [62]: # --- 3. Transform Data to WOE Values ---

def replace_with_woe(df, woe_df_list, feature_map):
    df_woe_final = df.copy()
    for col_name, woe_df in woe_df_list.items():
        woe_map = dict(zip(woe_df['Value'], woe_df['WoE']))
        binned_col = feature_map[col_name]
        df_woe_final[col_name + '_woe'] = df_woe_final[binned_col].map(woe_map)
    return df_woe_final

# Dictionary of your calculated WOE dataframes
woe_dataframes = {
    'Home_Ownership': woe_home,
    'Income': woe_income,
    'Emp_Length': woe_emp,
    'Loan_Amnt': woe_amnt,
    'Loan_Percent_Income': woe_lpi
}

# Mapping: {Feature Name in woe_dataframes : Column Name in df}
feature_map = {
    'Home_Ownership': 'Home_Ownership',
    'Income': 'income_bin',
    'Emp_Length': 'Emp_Length_bin',
    'Loan_Amnt': 'Loan_Amnt_bin',
    'Loan_Percent_Income': 'Loan_Percent_Income_bin'
}

# Transform
final_train_df = replace_with_woe(df, woe_dataframes, feature_map)

# Show the result
print(final_train_df.filter(like='_woe').head())

```

	Home_Ownership_woe	Income_woe	Emp_Length_woe	Loan_Amnt_woe	\
0	-0.502794	0.226724	-0.583251	-0.472078	
1	1.240379	-1.005062	0.116352	0.051321	
2	0.663067	-1.005062	-0.323016	0.288426	
3	-0.502794	0.493859	0.116352	-0.472078	
4	-0.502794	0.226724	0.295783	-0.472078	

	Loan_Percent_Income_woe
0	-1.39373
1	0.67320
2	-1.39373
3	-1.39373
4	-1.39373

```
In [63]: import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, roc_curve

# 1. Define X (Features) and y (Target)
# We only use the WOE columns we just created.
# Note: We EXCLUDE 'Loan_Int_Rate' (Leakage) and 'Age' (Weak)
feature_cols = [
    'Home_Ownership_woe',
    'Income_woe',
    'Emp_Length_woe',
    'Loan_Amnt_woe',
    'Loan_Percent_Income_woe' # Assuming you added this in the previous step
]

X = final_train_df[feature_cols]
y = final_train_df['Loan_Status']

# 2. Train-Test Split (80% Train, 20% Test)
# random_state=42 ensures we get the same split every time we run it
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 3. Train Logistic Regression Model (using Statsmodels)
# Statsmodels requires us to manually add a constant (Intercept)
X_train_const = sm.add_constant(X_train)
logit_model = sm.Logit(y_train, X_train_const)
result = logit_model.fit()

# 4. Print Model Summary
# Look for P>|z|. If value is < 0.05, the variable is significant.
print(result.summary())
```

Optimization terminated successfully.

Current function value: 0.418453

Iterations 6

Logit Regression Results

```
=====
Dep. Variable:          Loan_Status    No. Observations:          26064
Model:                  Logit          Df Residuals:              26058
Method:                 MLE           Df Model:                  5
Date:                  Sun, 21 Dec 2025 Pseudo R-squ.:             0.2006
Time:                  18:09:11       Log-Likelihood:            -10907.
converged:              True          LL-Null:                  -13643.
Covariance Type:       nonrobust      LLR p-value:              0.000
=====
```

```
=====
                                coef    std err          z      P>|z|      [0.025
0.975]
-----
const                -1.3022      0.018    -73.725     0.000    -1.337
-1.268
Home_Ownership_woe    -0.8839      0.029   -30.660     0.000    -0.940
-0.827
Income_woe            -0.8017      0.031   -25.537     0.000    -0.863
-0.740
Emp_Length_woe        -0.3238      0.072    -4.508     0.000    -0.465
-0.183
Loan_Amnt_woe         -1.2027      0.082   -14.690     0.000    -1.363
-1.042
Loan_Percent_Income_woe -0.7270      0.023   -31.427     0.000    -0.772
-0.682
=====
```

```
In [64]: # Predict on the Test Set
X_test_const = sm.add_constant(X_test)
y_pred_prob = result.predict(X_test_const)

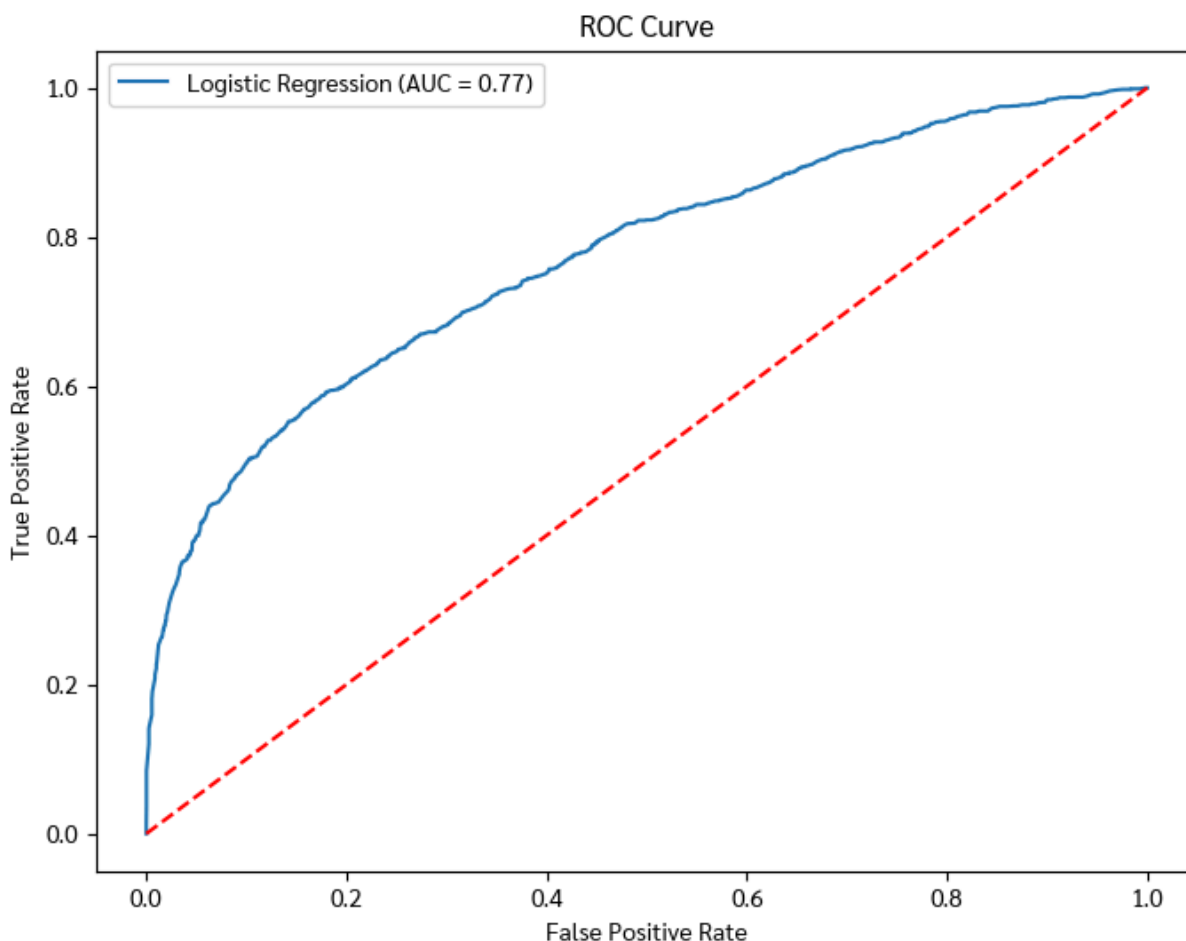
# Calculate AUC (Area Under Curve)
auc = roc_auc_score(y_test, y_pred_prob)
gini = 2 * auc - 1

print(f"AUC Score: {auc:.4f}")
print(f"Gini Coefficient: {gini:.4f}")

# Plot ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'Logistic Regression (AUC = {auc:.2f})')
plt.plot([0, 1], [0, 1], 'r--') # Random guess line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```

AUC Score: 0.7719

Gini Coefficient: 0.5437



Interpreting the results

3. Which factors drive the risk? (Coefficients)

The "Coef" column tells us how heavily the model weighs each factor. Since all variables are on the same WOE scale, we can roughly compare their magnitude:

- Loan Amount (-1.363): This is your strongest predictor. The size of the loan relative to the specific "bin" it falls into is the biggest driver of the final score.
- Home Ownership (-0.884): Whether a customer Rents, Owns, or has a Mortgage is the second most critical factor.
- Income (-0.802): Annual income power is the third driver.
- Loan % of Income (-0.727): The debt burden ratio is significant but slightly less impactful than the raw loan amount in this specific model.
- Employment Length (-0.465): This is the weakest predictor of the group (as we suspected from the lower IV earlier), but it is still statistically significant, so it is worth

keeping.

ScoreCard

The Scoring Logic

$$Score = Offset + Factor \times \ln(Odds)$$

```
In [65]: # --- 1. Settings ---
TARGET_SCORE = 600
TARGET_ODDS = 50
PDO = 20

# Calculate Factor and Offset
factor = PDO / np.log(2)
offset = TARGET_SCORE - (factor * np.log(TARGET_ODDS))

print(f"Factor: {factor:.2f}")
print(f"Offset: {offset:.2f}")

# --- 2. Extract Coefficients ---
# Get the intercept (alpha) and feature coefficients (betas) from your model
intercept = result.params['const']
coeffs = result.params.drop('const')

# --- 3. Calculate Base Score ---
# Since your model predicts Log(Bad/Good), and we want Score ~ Log(Good/Bad),
# we negate the equation.
# Base Score = Offset + Factor * (-Intercept)
base_score = offset + (factor * -intercept)

print(f"Base Score (Intercept Points): {base_score:.0f}")

# --- 4. Generate Scorecard Table ---
scorecard = []

# List of your features and their corresponding original WOE dataframes
# IMPORTANT: Ensure 'woe_dataframes' dictionary (from previous steps) is available
features_list = {
    'Home_Ownership_woe': ('Home_Ownership', woe_home),
    'Income_woe': ('Income', woe_income),
    'Emp_Length_woe': ('Emp_Length', woe_emp),
    'Loan_Amnt_woe': ('Loan_Amnt', woe_amnt),
    'Loan_Percent_Income_woe': ('Loan_Percent_Income', woe_lpi)
}

for col_model, (col_original, df_woe) in features_list.items():
    # Get the coefficient for this feature
    beta = coeffs[col_model]

    # Iterate through the bins in the WOE dataframe
    for index, row in df_woe.iterrows():
        woe_val = row['WoE']
```

```
bin_label = row['Value']

# Calculate Score Point for this bin
# Point = (-Factor) * (Coefficient) * (WOE)
points = (-factor) * beta * woe_val

scorecard.append({
    'Feature': col_original,
    'Bin': bin_label,
    'WoE': woe_val,
    'Points': round(points) # Round to nearest integer
})

# Create DataFrame
df_scorecard = pd.DataFrame(scorecard)

# --- 5. Display Final Scorecard ---
print("\n--- FINAL CREDIT SCORECARD ---")
print(f"Base Score: {round(base_score)}")
display(df_scorecard)
```

Factor: 28.85

Offset: 487.12

Base Score (Intercept Points): 525

--- FINAL CREDIT SCORECARD ---

Base Score: 525

	Feature	Bin	WoE	Points
0	Home_Ownership	RENT	-0.502794	-13
1	Home_Ownership	OWN	1.240379	32
2	Home_Ownership	MORTGAGE	0.663067	17
3	Home_Ownership	OTHER	-0.468841	-12
4	Income	(49000.0, 63000.0]	0.226724	5
5	Income	(3999.0, 35000.0]	-1.005062	-23
6	Income	(63000.0, 86000.0]	0.493859	11
7	Income	(86000.0, 6000000.0]	1.021115	24
8	Income	(35000.0, 49000.0]	-0.093293	-2
9	Emp_Length	>40	-0.583251	-5
10	Emp_Length	4-6	0.116352	1
11	Emp_Length	<2	-0.323016	-3
12	Emp_Length	8-10	0.295783	3
13	Emp_Length	2-4	-0.070289	-1
14	Emp_Length	6-8	0.189433	2
15	Emp_Length	10-40	0.352380	3
16	Emp_Length	Missing	-0.499940	-5
17	Loan_Amnt	(14500.0, 35000.0]	-0.472078	-16
18	Loan_Amnt	(499.0, 4400.0]	0.051321	2
19	Loan_Amnt	(4400.0, 6750.0]	0.288426	10
20	Loan_Amnt	(6750.0, 10000.0]	0.174890	6
21	Loan_Amnt	(10000.0, 14500.0]	-0.000502	0
22	Loan_Percent_Income	(0.25, 0.83]	-1.393730	-29
23	Loan_Percent_Income	(0.08, 0.12]	0.673200	14
24	Loan_Percent_Income	(0.18, 0.25]	0.164674	3
25	Loan_Percent_Income	(0.12, 0.18]	0.444861	9
26	Loan_Percent_Income	(-0.01, 0.08]	0.773085	16