nisarvati initial prototype 4547411 · 12 hours ago

3554 lines (3554 loc) · 1.89 MB

In [4]:

```python
# CELL 1: Import Libraries

import warnings
warnings.filterwarnings('ignore')

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, RandomizedSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier

from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    roc_auc_score, roc_curve, confusion_matrix,
    classification_report, precision_recall_curve
)

import shap

plt.style.use('seaborn-v0_8-whitegrid')
sns.set_palette("husl")

RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)

print("Libraries imported")
print("Focus: Alternative Credit Scoring for Unbanked Users")
```

```
Libraries imported
Focus: Alternative Credit Scoring for Unbanked Users
```

In [5]:

```python
# CELL 2: Load Data

df = pd.read_csv('cs-training.csv')

print("Data loaded: Give Me Some Credit Dataset")
print(f"Shape: {df.shape}")
print(f"\nColumns: {df.columns.tolist()}")
print(f"\nFirst 3 rows:")
print(df.head(3))
```

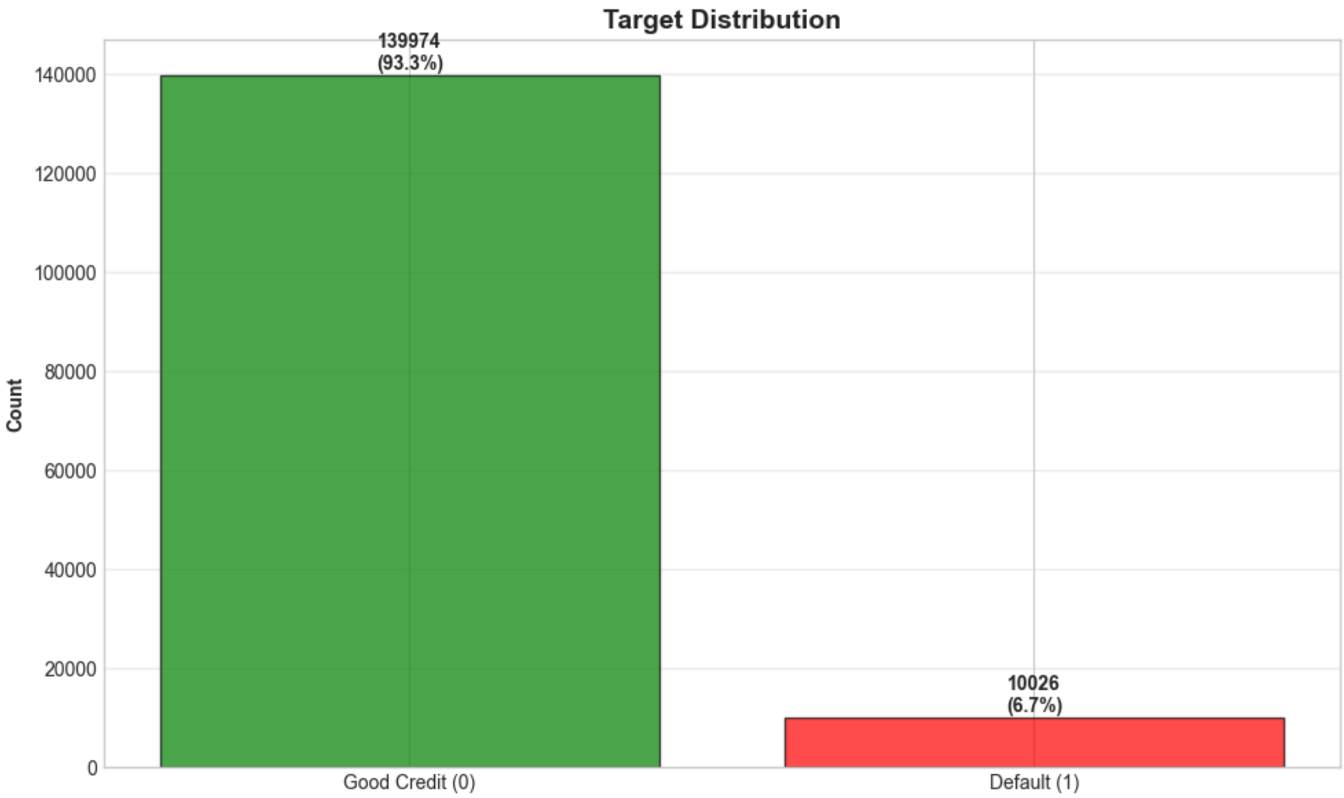
```
Data loaded: Give Me Some Credit Dataset
Shape: (150000, 12)

Columns: ['Unnamed: 0', 'SeriousDlqin2yrs', 'RevolvingUtilizationOfUnsecuredLines', 'age', 'NumberOfTime30-5
9DaysPastDueNotWorse', 'DebtRatio', 'MonthlyIncome', 'NumberOfOpenCreditLinesAndLoans', 'NumberOfTimes90Days
Late', 'NumberRealEstateLoansOrLines', 'NumberOfTime60-89DaysPastDueNotWorse', 'NumberOfDependents']

First 3 rows:
   Unnamed: 0  SeriousDlqin2yrs  RevolvingUtilizationOfUnsecuredLines  age  \
0           1                 1                              0.766127   45
1           2                 0                              0.957151   40
2           3                 0                              0.658180   38

   NumberOfTime30-59DaysPastDueNotWorse  DebtRatio  MonthlyIncome  \
0                                     2   0.802982         9120.0
1                                     0   0.121876         2600.0
2                                     1   0.085113         3042.0

   NumberOfOpenCreditLinesAndLoans  NumberOfTimes90DaysLate  \
0                               13                        0
1                                4                        0
2                                2                        1

   NumberRealEstateLoansOrLines  NumberOfTime60-89DaysPastDueNotWorse  \
0                             6                                     0
1                             0                                     0
2                             0                                     0

   NumberOfDependents
0                 2.0
1                 1.0
2                 0.0
```

In [6]:

```python
# CELL 3: Create Target Variable
```

```python
print("Creating target variable...")

y = df['SeriousDlqin2yrs'].values

print(f"\nTarget distribution:")
print(f"Good Credit (0): {(y==0).sum()} ({(y==0).mean()*100:.1f}%)")
print(f"Default (1): {(y==1).sum()} ({(y==1).mean()*100:.1f}%)")
print(f"Default rate: {y.mean():.2%}")

plt.figure(figsize=(10, 6))
counts = pd.Series(y).value_counts().sort_index()
plt.bar(['Good Credit (0)', 'Default (1)'], counts.values,
        color=['green', 'red'], alpha=0.7, edgecolor='black')
plt.title('Target Distribution', fontweight='bold', fontsize=14)
plt.ylabel('Count', fontweight='bold')
plt.grid(axis='y', alpha=0.3)

for i, v in enumerate(counts.values):
    pct = v / len(y) * 100
    plt.text(i, v + 1000, f'{v}\n({pct:.1f}%)', ha='center', fontweight='bold')

plt.tight_layout()
plt.show()
```

```
Creating target variable...

Target distribution:
Good Credit (0): 139974 (93.3%)
Default (1): 10026 (6.7%)
Default rate: 6.68%
```



In [7]:

```python
# CELL 4: Create Feature Matrix

print("Creating feature matrix...")

X = df.drop(columns=['SeriousDlqin2yrs', 'Unnamed: 0'], errors='ignore')

print(f"\nFeature matrix: {X.shape}")
print(f"Features: {X.columns.tolist()}")
```

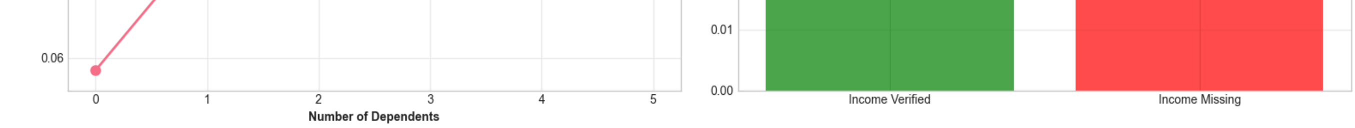
```
Creating feature matrix...

Feature matrix: (150000, 10)
Features: ['RevolvingUtilizationOfUnsecuredLines', 'age', 'NumberOfTime30-59DaysPastDueNotWorse', 'DebtRati
o', 'MonthlyIncome', 'NumberOfOpenCreditLinesAndLoans', 'NumberOfTimes90DaysLate', 'NumberRealEstateLoansOrL
ines', 'NumberOfTime60-89DaysPastDueNotWorse', 'NumberOfDependents']
```

In [8]:

```python
# CELL 5: Exploratory Data Analysis - Target Analysis

print("EXPLORATORY DATA ANALYSIS")
print("Part 1: Target Variable Analysis")

fig, axes = plt.subplots(2, 2, figsize=(16, 12))
```

```python
# 1. Target distribution
counts = df['SeriousDlqin2yrs'].value_counts()
axes[0, 0].bar(['Good (0)', 'Default (1)'], counts.values, color=['green', 'red'], alpha=0.7)
axes[0, 0].set_ylabel('Count', fontweight='bold')
axes[0, 0].set_title('Target Distribution', fontweight='bold')
axes[0, 0].grid(axis='y', alpha=0.3)
for i, v in enumerate(counts.values):
    axes[0, 0].text(i, v+1000, f'{v:,}', ha='center', fontweight='bold')

# 2. Default rate by age groups
df['age_group'] = pd.cut(df['age'], bins=[0, 25, 35, 45, 55, 65, 100],
                          labels=['18-25', '26-35', '36-45', '46-55', '56-65', '65+'])
default_by_age = df.groupby('age_group')['SeriousDlqin2yrs'].agg(['mean', 'count'])
axes[0, 1].bar(range(len(default_by_age)), default_by_age['mean'], alpha=0.7, color='coral')
axes[0, 1].set_xticks(range(len(default_by_age)))
axes[0, 1].set_xticklabels(default_by_age.index, rotation=45)
axes[0, 1].set_ylabel('Default Rate', fontweight='bold')
axes[0, 1].set_title('Default Rate by Age Group', fontweight='bold')
axes[0, 1].grid(axis='y', alpha=0.3)
for i, v in enumerate(default_by_age['mean']):
    axes[0, 1].text(i, v+0.005, f'{v:.1%}', ha='center', fontweight='bold')

# 3. Default by number of dependents
default_by_dep = df.groupby('NumberOfDependents')['SeriousDlqin2yrs'].mean().head(6)
axes[1, 0].plot(default_by_dep.index, default_by_dep.values, marker='o', linewidth=2, markersize=8)
axes[1, 0].set_xlabel('Number of Dependents', fontweight='bold')
axes[1, 0].set_ylabel('Default Rate', fontweight='bold')
axes[1, 0].set_title('Default Rate by Number of Dependents', fontweight='bold')
axes[1, 0].grid(alpha=0.3)

# 4. Default by income availability
income_status = df.groupby(df['MonthlyIncome'].isna())['SeriousDlqin2yrs'].mean()
labels = ['Income Verified', 'Income Missing']
axes[1, 1].bar(labels, income_status.values, color=['green', 'red'], alpha=0.7)
axes[1, 1].set_ylabel('Default Rate', fontweight='bold')
axes[1, 1].set_title('Default Rate by Income Verification', fontweight='bold')
axes[1, 1].grid(axis='y', alpha=0.3)
for i, v in enumerate(income_status.values):
    axes[1, 1].text(i, v+0.005, f'{v:.1%}', ha='center', fontweight='bold')

plt.tight_layout()
plt.show()

print("\nKey Insights:")
print(f"  - Younger borrowers (18-25) have higher default rates")
print(f"  - Missing income data correlates with {income_status.values[1]:.1%} default rate")
print(f"  - Default rate varies by family size")

df.drop('age_group', axis=1, inplace=True)
```

```
EXPLORATORY DATA ANALYSIS
Part 1: Target Variable Analysis
```

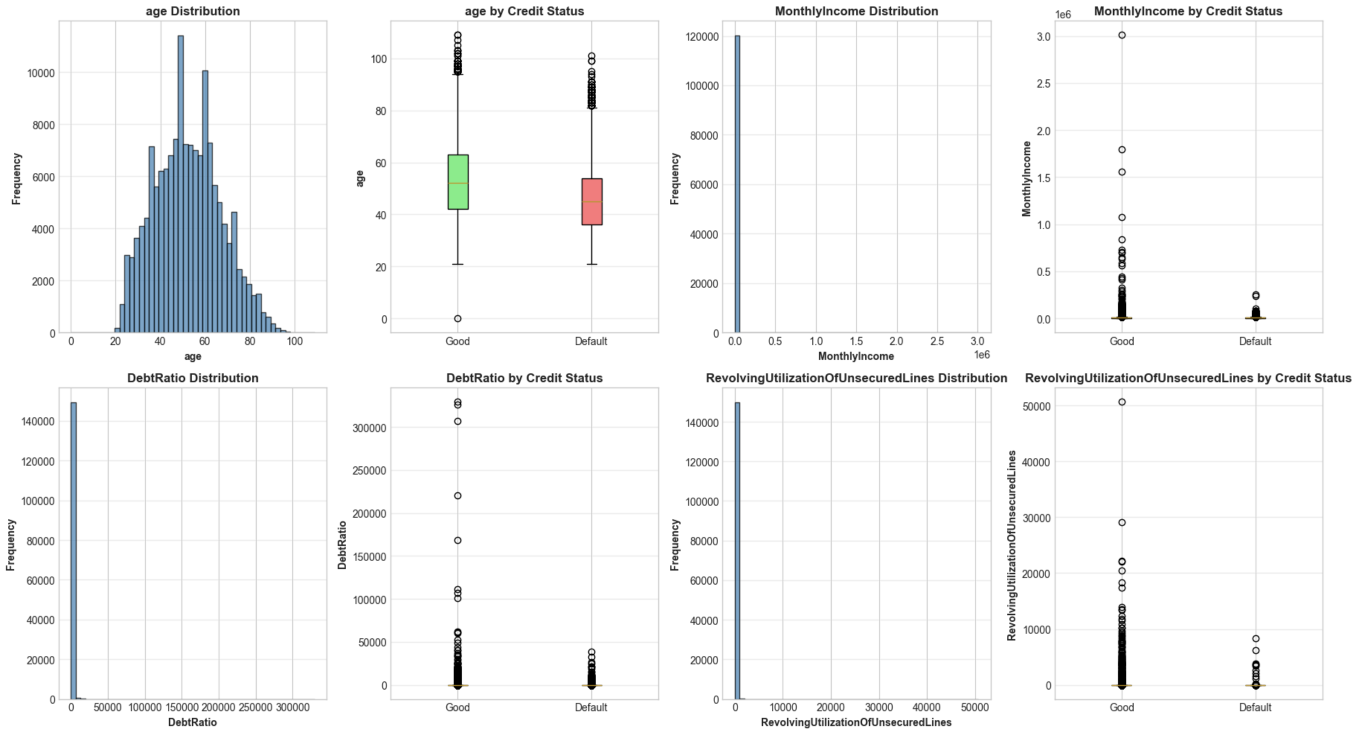



```
Key Insights:
  - Younger borrowers (18-25) have higher default rates
  - Missing income data correlates with 5.6% default rate
  - Default rate varies by family size
```

In [9]:

```python
# CELL 6: EDA - Feature Distributions

print("Part 2: Feature Distributions")

features_to_plot = ['age', 'MonthlyIncome', 'DebtRatio', 'RevolvingUtilizationOfUnsecuredLines']

fig, axes = plt.subplots(2, 4, figsize=(18, 10))

for idx, feature in enumerate(features_to_plot):
    row = idx // 2

    # Distribution plot
    col = (idx % 2) * 2
    axes[row, col].hist(df[feature].dropna(), bins=50, edgecolor='black', alpha=0.7, color='steelblue')
    axes[row, col].set_xlabel(feature, fontweight='bold')
    axes[row, col].set_ylabel('Frequency', fontweight='bold')
    axes[row, col].set_title(f'{feature} Distribution', fontweight='bold')
    axes[row, col].grid(axis='y', alpha=0.3)

    # Boxplot by target
    col = (idx % 2) * 2 + 1
    data_to_plot = [df[df['SeriousDlqin2yrs']==0][feature].dropna(),
                    df[df['SeriousDlqin2yrs']==1][feature].dropna()]
    bp = axes[row, col].boxplot(data_to_plot, labels=['Good', 'Default'], patch_artist=True)
    for patch, color in zip(bp['boxes'], ['lightgreen', 'lightcoral']):
        patch.set_facecolor(color)
    axes[row, col].set_ylabel(feature, fontweight='bold')
    axes[row, col].set_title(f'{feature} by Credit Status', fontweight='bold')
    axes[row, col].grid(axis='y', alpha=0.3)

plt.tight_layout()
plt.show()

print("\nDistribution Statistics:")
for feature in features_to_plot:
    print(f"\n{feature}:")
    print(f"  Mean: {df[feature].mean():.2f}")
    print(f"  Median: {df[feature].median():.2f}")
    print(f"  Std: {df[feature].std():.2f}")
    print(f"  Missing: {df[feature].isna().sum()} ({df[feature].isna().sum()/len(df)*100:.1f}%)")
```

```
Part 2: Feature Distributions
```




```
Distribution Statistics:

age:
  Mean: 52.30
  Median: 52.00
  Std: 14.77
```

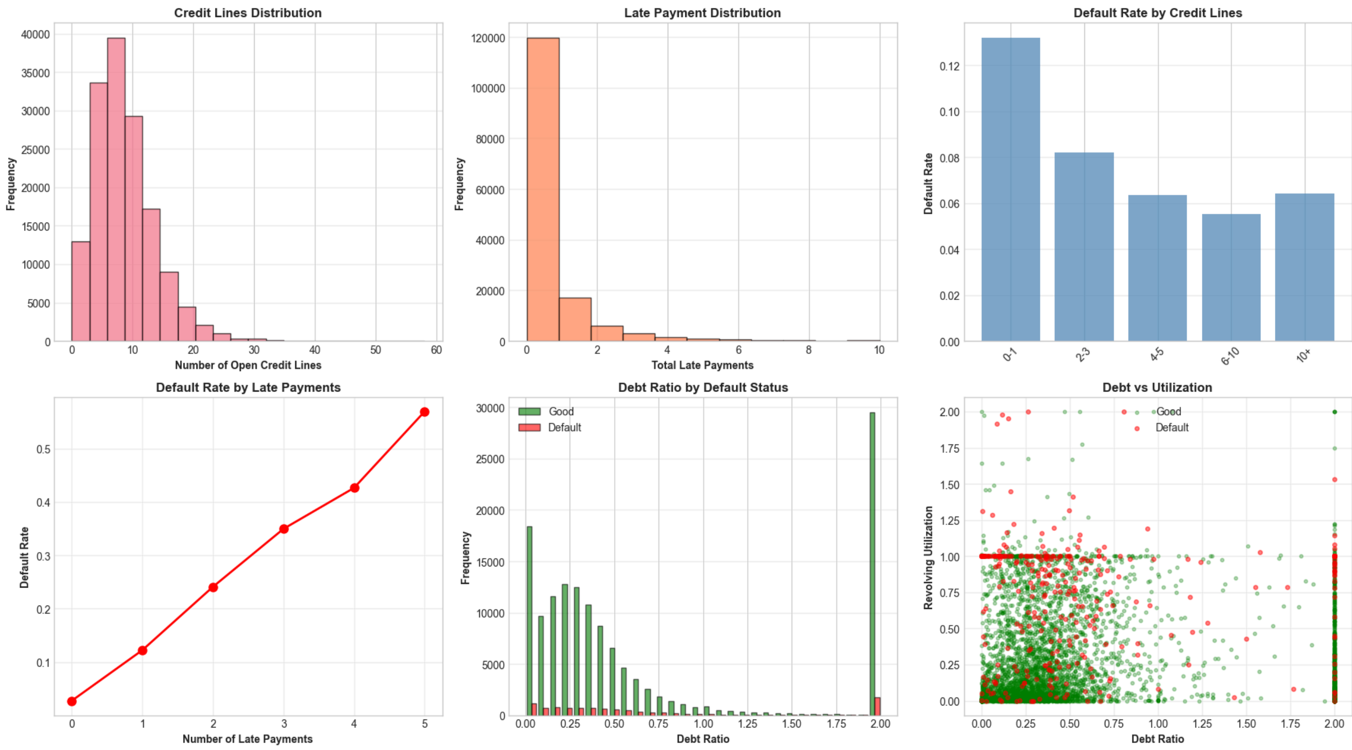
```
  Missing: 0 (0.0%)

MonthlyIncome:
  Mean: 6670.22
  Median: 5400.00
  Std: 14384.67
  Missing: 29731 (19.8%)

DebtRatio:
  Mean: 353.01
  Median: 0.37
  Std: 2037.82
  Missing: 0 (0.0%)

RevolvingUtilizationOfUnsecuredLines:
  Mean: 6.05
  Median: 0.15
  Std: 249.76
  Missing: 0 (0.0%)
```

In [10]:

```python
# CELL 7: EDA - Credit Behavior Analysis

print("Part 3: Credit Behavior Analysis")

fig, axes = plt.subplots(2, 3, figsize=(18, 10))

# 1. Credit lines distribution
axes[0, 0].hist(df['NumberOfOpenCreditLinesAndLoans'], bins=20, edgecolor='black', alpha=0.7)
axes[0, 0].set_xlabel('Number of Open Credit Lines', fontweight='bold')
axes[0, 0].set_ylabel('Frequency', fontweight='bold')
axes[0, 0].set_title('Credit Lines Distribution', fontweight='bold')
axes[0, 0].grid(axis='y', alpha=0.3)

# 2. Late payments distribution
late_payment_cols = ['NumberOfTime30-59DaysPastDueNotWorse',
                     'NumberOfTime60-89DaysPastDueNotWorse',
                     'NumberOfTimes90DaysLate']
df['total_late_payments'] = df[late_payment_cols].sum(axis=1)
axes[0, 1].hist(df['total_late_payments'].clip(0, 10), bins=11, edgecolor='black', alpha=0.7, color='coral'
axes[0, 1].set_xlabel('Total Late Payments', fontweight='bold')
axes[0, 1].set_ylabel('Frequency', fontweight='bold')
axes[0, 1].set_title('Late Payment Distribution', fontweight='bold')
axes[0, 1].grid(axis='y', alpha=0.3)

# 3. Default rate by credit lines
credit_line_bins = [0, 1, 3, 5, 10, 100]
df['credit_line_group'] = pd.cut(df['NumberOfOpenCreditLinesAndLoans'], bins=credit_line_bins)
default_by_credit = df.groupby('credit_line_group')['SeriousDlqin2yrs'].mean()
axes[0, 2].bar(range(len(default_by_credit)), default_by_credit.values, alpha=0.7, color='steelblue')
axes[0, 2].set_xticks(range(len(default_by_credit)))
axes[0, 2].set_xticklabels(['0-1', '2-3', '4-5', '6-10', '10+'], rotation=45)
axes[0, 2].set_ylabel('Default Rate', fontweight='bold')
axes[0, 2].set_title('Default Rate by Credit Lines', fontweight='bold')
axes[0, 2].grid(axis='y', alpha=0.3)

# 4. Default rate by late payments
default_by_late = df.groupby(df['total_late_payments'].clip(0, 5))['SeriousDlqin2yrs'].mean()
axes[1, 0].plot(default_by_late.index, default_by_late.values, marker='o', linewidth=2, markersize=8, color
axes[1, 0].set_xlabel('Number of Late Payments', fontweight='bold')
axes[1, 0].set_ylabel('Default Rate', fontweight='bold')
axes[1, 0].set_title('Default Rate by Late Payments', fontweight='bold')
axes[1, 0].grid(alpha=0.3)

# 5. Debt ratio distribution by default
data_good = df[df['SeriousDlqin2yrs']==0]['DebtRatio'].clip(0, 2)
data_default = df[df['SeriousDlqin2yrs']==1]['DebtRatio'].clip(0, 2)
axes[1, 1].hist([data_good, data_default], bins=30, label=['Good', 'Default'],
                color=['green', 'red'], alpha=0.6, edgecolor='black')
axes[1, 1].set_xlabel('Debt Ratio', fontweight='bold')
axes[1, 1].set_ylabel('Frequency', fontweight='bold')
axes[1, 1].set_title('Debt Ratio by Default Status', fontweight='bold')
axes[1, 1].legend()
axes[1, 1].grid(axis='y', alpha=0.3)

# 6. Utilization vs Debt Ratio scatter
sample = df.sample(min(5000, len(df)), random_state=RANDOM_SEED)
good = sample[sample['SeriousDlqin2yrs']==0]
default = sample[sample['SeriousDlqin2yrs']==1]
axes[1, 2].scatter(good['DebtRatio'].clip(0, 2),
                   good['RevolvingUtilizationOfUnsecuredLines'].clip(0, 2),
                   alpha=0.3, s=10, c='green', label='Good')
axes[1, 2].scatter(default['DebtRatio'].clip(0, 2),
                   default['RevolvingUtilizationOfUnsecuredLines'].clip(0, 2),
                   alpha=0.5, s=15, c='red', label='Default')
axes[1, 2].set_xlabel('Debt Ratio', fontweight='bold')
axes[1, 2].set_ylabel('Revolving Utilization', fontweight='bold')
axes[1, 2].set_title('Debt vs Utilization', fontweight='bold')
```

```
axes[1, 2].set_title('Debt vs Utilization', fontweight='bold')
axes[1, 2].legend()
axes[1, 2].grid(alpha=0.3)

plt.tight_layout()
plt.show()

df.drop(['total_late_payments', 'credit_line_group'], axis=1, inplace=True, errors='ignore')

print("\nCredit Behavior Insights:")
print(f"  – {(df['NumberOfOpenCreditLinesAndLoans']==0).sum():,} users have NO credit lines")
print(f"  – Late payments strongly correlate with default")
print(f"  – High debt ratio and utilization = higher risk")
```

```
Part 3: Credit Behavior Analysis
```

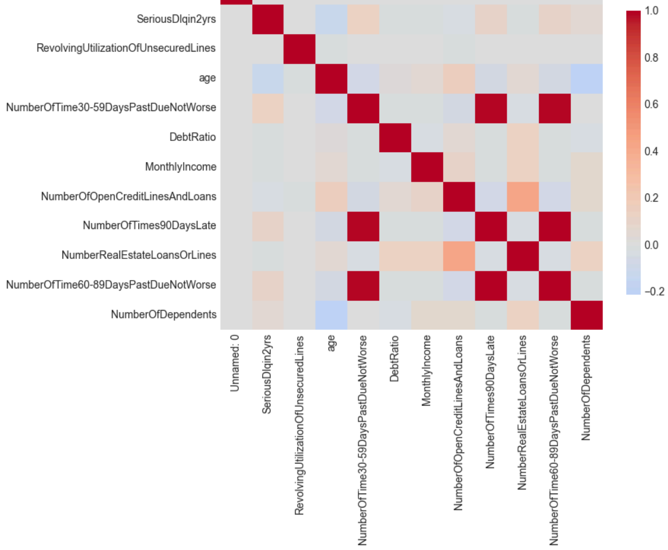


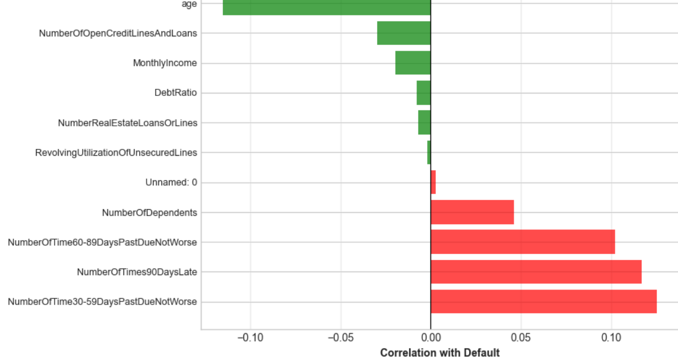

```
Credit Behavior Insights:
  – 1,888 users have NO credit lines
  – Late payments strongly correlate with default
  – High debt ratio and utilization = higher risk
```

In [11]:

```
# CELL 8: EDA – Correlation Analysis

print("Part 4: Correlation Analysis")

# Select numeric columns
numeric_cols = df.select_dtypes(include=[np.number]).columns
corr_data = df[numeric_cols].corr()

fig, axes = plt.subplots(1, 2, figsize=(18, 8))

# Full correlation heatmap
sns.heatmap(corr_data, annot=False, cmap='coolwarm', center=0,
            cbar_kws={'shrink': 0.8}, ax=axes[0])
axes[0].set_title('Feature Correlation Heatmap', fontweight='bold', fontsize=14)

# Correlation with target
target_corr = corr_data['SeriousDlqin2yrs'].sort_values(ascending=False).drop('SeriousDlqin2yrs')
colors = ['green' if x < 0 else 'red' for x in target_corr.values]
axes[1].barh(range(len(target_corr)), target_corr.values, color=colors, alpha=0.7)
axes[1].set_yticks(range(len(target_corr)))
axes[1].set_yticklabels(target_corr.index, fontsize=9)
axes[1].set_xlabel('Correlation with Default', fontweight='bold')
axes[1].set_title('Feature Correlation with Target', fontweight='bold', fontsize=14)
axes[1].axvline(0, color='black', linewidth=0.8)
axes[1].grid(axis='x', alpha=0.3)

plt.tight_layout()
plt.show()

print("\nTop 5 Positive Correlations with Default:")
print(target_corr.head().to_string())

print("\nTop 5 Negative Correlations with Default:")
print(target_corr.tail().to_string())
```

```
Part 4: Correlation Analysis
```






```
Top 5 Positive Correlations with Default:
NumberOfTime30-59DaysPastDueNotWorse    0.125587
NumberOfTimes90DaysLate                 0.117175
NumberOfTime60-89DaysPastDueNotWorse    0.102261
NumberOfDependents                      0.046048
Unnamed: 0                              0.002801

Top 5 Negative Correlations with Default:
NumberRealEstateLoansOrLines      -0.007038
DebtRatio                         -0.007602
MonthlyIncome                     -0.019746
NumberOfOpenCreditLinesAndLoans   -0.029669
age                               -0.115386
```

In [12]:

```python
# CELL 9: EDA - Summary Statistics

print("Part 5: Summary Statistics")

print("\nDataset Overview:")
print(f"  Total records: {len(df):,}")
print(f"  Features: {len(df.columns)}")
print(f"  Default rate: {df['SeriousDlqin2yrs'].mean():.2%}")

print("\nKey Statistics by Default Status:")
summary_stats = df.groupby('SeriousDlqin2yrs').agg({
    'age': ['mean', 'median'],
    'MonthlyIncome': ['mean', 'median'],
    'DebtRatio': ['mean', 'median'],
    'NumberOfOpenCreditLinesAndLoans': ['mean', 'median'],
    'NumberOfTime30-59DaysPastDueNotWorse': 'mean',
    'NumberOfTimes90DaysLate': 'mean'
}).round(2)

print("\n" + summary_stats.to_string())

print("\nEDA COMPLETE")
print("Key findings will inform feature engineering")
```

```
Part 5: Summary Statistics

Dataset Overview:
  Total records: 150,000
  Features: 12
  Default rate: 6.68%

Key Statistics by Default Status:

                    age        MonthlyIncome         DebtRatio        NumberOfOpenCreditLinesAndLoans
NumberOfTime30-59DaysPastDueNotWorse NumberOfTimes90DaysLate
                   mean median          mean  median        mean median                            mean median
mean                    mean
SeriousDlqin2yrs
0                 52.75   52.0       6747.84  5466.0      357.15   0.36                            8.49    8.0
0.28                    0.14
1                 45.93   45.0       5630.83  4500.0      295.12   0.43                            7.88    7.0
2.39                    2.09

EDA COMPLETE
Key findings will inform feature engineering
```

In [13]:

```python
# CELL 6: Data Cleaning - Handle Missing Values

print("DATA CLEANING: HANDLING MISSING VALUES")

print("\nMissing values per column:")
```

```
for col in X.columns:
    missing = X[col].isnull().sum()
    if missing > 0:
        pct = missing / len(X) * 100
        print(f"  {col}: {missing:,} ({pct:.2f}%)")

# Strategy: Use median for numeric columns
print("\nImputation strategy: Median for all numeric features")

for col in X.columns:
    if X[col].isnull().sum() > 0:
        X[col].fillna(X[col].median(), inplace=True)
        print(f"  Filled {col} with median: {X[col].median():.2f}")

print("\nVerification: Missing values after imputation:")
print(X.isnull().sum().sum())
```

```
DATA CLEANING: HANDLING MISSING VALUES

Missing values per column:
  MonthlyIncome: 29,731 (19.82%)
  NumberOfDependents: 3,924 (2.62%)

Imputation strategy: Median for all numeric features
  Filled MonthlyIncome with median: 5400.00
  Filled NumberOfDependents with median: 0.00

Verification: Missing values after imputation:
33655
```

In [14]:

```
# CELL 7: Data Cleaning - Handle Outliers

print("DATA CLEANING: HANDLING OUTLIERS")

print("\nOutlier detection and capping at 1st and 99th percentiles")

numeric_cols = X.select_dtypes(include=[np.number]).columns

outlier_summary = []
for col in numeric_cols:
    q1 = X[col].quantile(0.01)
    q99 = X[col].quantile(0.99)

    outliers_low = (X[col] < q1).sum()
    outliers_high = (X[col] > q99).sum()
    total_outliers = outliers_low + outliers_high

    if total_outliers > 0:
        outlier_summary.append({
            'Feature': col,
            'Outliers': total_outliers,
            'Percentage': f"{total_outliers/len(X)*100:.2f}%"
        })
        X[col] = X[col].clip(q1, q99)

print("\nOutliers detected and capped:")
if outlier_summary:
    print(pd.DataFrame(outlier_summary).to_string(index=False))
else:
    print("No significant outliers found")

print("\nOutlier handling complete")
```

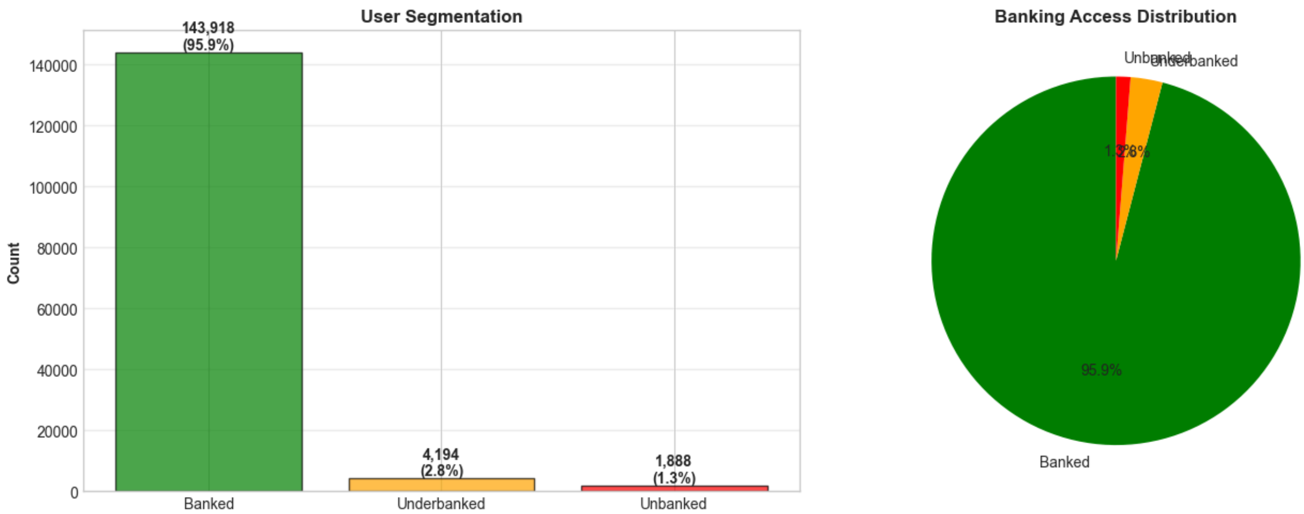
```
DATA CLEANING: HANDLING OUTLIERS

Outlier detection and capping at 1st and 99th percentiles

Outliers detected and capped:
                             Feature  Outliers Percentage
RevolvingUtilizationOfUnsecuredLines      1500      1.00%
                                 age      2535      1.69%
NumberOfTime30-59DaysPastDueNotWorse       850      0.57%
                           DebtRatio      1500      1.00%
                       MonthlyIncome      1168      0.78%
     NumberOfOpenCreditLinesAndLoans      1476      0.98%
             NumberOfTimes90DaysLate       873      0.58%
        NumberRealEstateLoansOrLines      1482      0.99%
NumberOfTime60-89DaysPastDueNotWorse       755      0.50%
                  NumberOfDependents       991      0.66%

Outlier handling complete
```

In [15]:

```
# CELL 8: Identify Unbanked Users

print("IDENTIFYING UNBANKED USERS")
```

```
# Definition: Unbanked = No credit lines AND no real estate loans
X['unbanked_proxy'] = (
    (X['NumberOfOpenCreditLinesAndLoans'] == 0) &
    (X['NumberRealEstateLoansOrLines'] == 0)
).astype(int)

# Underbanked: Minimal credit history
X['underbanked_proxy'] = (
    (X['NumberOfOpenCreditLinesAndLoans'] <= 1) &
    (X['NumberRealEstateLoansOrLines'] == 0) &
    (X['unbanked_proxy'] == 0)
).astype(int)

# Traditional banking users
X['banked'] = (
    (X['unbanked_proxy'] == 0) &
    (X['underbanked_proxy'] == 0)
).astype(int)

unbanked_count = X['unbanked_proxy'].sum()
underbanked_count = X['underbanked_proxy'].sum()
banked_count = X['banked'].sum()

print(f"\nUser Segmentation:")
print(f"  Unbanked: {unbanked_count:,} ({unbanked_count/len(X)*100:.1f}%)")
print(f"  Underbanked: {underbanked_count:,} ({underbanked_count/len(X)*100:.1f}%)")
print(f"  Banked: {banked_count:,} ({banked_count/len(X)*100:.1f}%)")

# Visualize
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

categories = ['Banked', 'Underbanked', 'Unbanked']
values = [banked_count, underbanked_count, unbanked_count]
colors = ['green', 'orange', 'red']

axes[0].bar(categories, values, color=colors, alpha=0.7, edgecolor='black')
axes[0].set_ylabel('Count', fontweight='bold')
axes[0].set_title('User Segmentation', fontweight='bold')
axes[0].grid(axis='y', alpha=0.3)

for i, v in enumerate(values):
    pct = v / len(X) * 100
    axes[0].text(i, v + 1000, f'{v:,}\n({pct:.1f}%)', ha='center', fontweight='bold')

axes[1].pie(values, labels=categories, colors=colors,
            autopct='%1.1f%%', startangle=90)
axes[1].set_title('Banking Access Distribution', fontweight='bold')

plt.tight_layout()
plt.show()

print(f"\nKEY INSIGHT: {(unbanked_count + underbanked_count)/len(X)*100:.1f}% lack traditional credit")
print("These users need alternative credit scoring methods")
```

```
IDENTIFYING UNBANKED USERS

User Segmentation:
  Unbanked: 1,888 (1.3%)
  Underbanked: 4,194 (2.8%)
  Banked: 143,918 (95.9%)
```




```
KEY INSIGHT: 4.1% lack traditional credit
These users need alternative credit scoring methods
```

In [16]:

```
# CELL 9: Alternative Credit Features

print("ENGINEERING ALTERNATIVE CREDIT FEATURES")
```

```python
print("Designed for users without traditional banking history")

# 1. Payment Discipline Score
X['payment_discipline_score'] = (
    1 - (X['NumberOfTime30-59DaysPastDueNotWorse'] +
         X['NumberOfTime60-89DaysPastDueNotWorse'] * 1.5 +
         X['NumberOfTimes90DaysLate'] * 2) / 20
).clip(0, 1)

X['perfect_payment_history'] = (
    (X['NumberOfTime30-59DaysPastDueNotWorse'] == 0) &
    (X['NumberOfTime60-89DaysPastDueNotWorse'] == 0) &
    (X['NumberOfTimes90DaysLate'] == 0)
).astype(int)

X['has_late_payments'] = (
    (X['NumberOfTime30-59DaysPastDueNotWorse'] > 0) |
    (X['NumberOfTime60-89DaysPastDueNotWorse'] > 0) |
    (X['NumberOfTimes90DaysLate'] > 0)
).astype(int)

# 2. Financial Stability Indicators
X['employment_years'] = (X['age'] - 18).clip(0, None)
X['working_age_prime'] = ((X['age'] >= 25) & (X['age'] <= 55)).astype(int)
X['young_adult'] = ((X['age'] >= 18) & (X['age'] < 30)).astype(int)
X['senior_borrower'] = (X['age'] > 60).astype(int)

# 3. Responsibility Indicators
X['has_dependents'] = (X['NumberOfDependents'] > 0).astype(int)
X['large_family'] = (X['NumberOfDependents'] > 3).astype(int)

# 4. Debt Management Capability
X['debt_to_income_risk'] = pd.cut(X['DebtRatio'],
                                   bins=[0, 0.3, 0.5, 0.7, np.inf],
                                   labels=[0, 1, 2, 3]).cat.codes

X['low_debt_burden'] = (X['DebtRatio'] < 0.3).astype(int)
X['moderate_debt'] = ((X['DebtRatio'] >= 0.3) & (X['DebtRatio'] < 0.5)).astype(int)
X['high_debt_burden'] = (X['DebtRatio'] >= 0.5).astype(int)

# 5. Credit Utilization Patterns
X['utilization_risk'] = pd.cut(X['RevolvingUtilizationOfUnsecuredLines'],
                                bins=[0, 0.3, 0.5, 0.8, np.inf],
                                labels=[0, 1, 2, 3]).cat.codes

X['healthy_utilization'] = (X['RevolvingUtilizationOfUnsecuredLines'] < 0.3).astype(int)
X['risky_utilization'] = (X['RevolvingUtilizationOfUnsecuredLines'] > 0.8).astype(int)

# 6. Income Verification
X['income_verified'] = (~X['MonthlyIncome'].isna()).astype(int)

# 7. Composite Scores for Unbanked
X['financial_inclusion_score'] = (
    0.30 * X['payment_discipline_score'] +
    0.20 * X['perfect_payment_history'] +
    0.20 * X['low_debt_burden'] +
    0.15 * X['working_age_prime'] +
    0.15 * (1 - X['has_late_payments'])
).round(3)

X['alternative_creditworthiness'] = (
    0.25 * X['payment_discipline_score'] +
    0.20 * (1 - X['high_debt_burden']) +
    0.20 * X['perfect_payment_history'] +
    0.20 * (X['employment_years'] / 40).clip(0, 1) +
    0.15 * X['income_verified']
).round(3)

# 8. Risk Flags
X['multiple_late_payments'] = (
    X['NumberOfTime30-59DaysPastDueNotWorse'] +
    X['NumberOfTime60-89DaysPastDueNotWorse'] +
    X['NumberOfTimes90DaysLate']
) > 2

X['severe_delinquency'] = (X['NumberOfTimes90DaysLate'] > 0).astype(int)

print(f"\nEngineered {X.shape[1] - 11} new features")
print(f"Total features: {X.shape[1]}")

print("\nKey Alternative Credit Features:")
print("  1. payment_discipline_score")
print("  2. financial_inclusion_score")
print("  3. alternative_creditworthiness")
print("  4. employment_years")
print("  5. debt_to_income_risk")
print("  6. utilization_risk")
```

```
print("\nThese features enable credit scoring without traditional banking data")
```

```
ENGINEERING ALTERNATIVE CREDIT FEATURES
Designed for users without traditional banking history

Engineered 23 new features
Total features: 34

Key Alternative Credit Features:
  1. payment_discipline_score
  2. financial_inclusion_score
  3. alternative_creditworthiness
  4. employment_years
  5. debt_to_income_risk
  6. utilization_risk

These features enable credit scoring without traditional banking data
```

In [17]:

```
# CELL 10: Train-Test Split

print("TRAIN-TEST SPLIT")

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

print(f"\nTrain set: {X_train.shape}")
print(f"  Good (0): {(y_train==0).sum():,} ({(y_train==0).mean()*100:.1f}%)")
print(f"  Default (1): {(y_train==1).sum():,} ({(y_train==1).mean()*100:.1f}%)")

print(f"\nTest set: {X_test.shape}")
print(f"  Good (0): {(y_test==0).sum():,} ({(y_test==0).mean()*100:.1f}%)")
print(f"  Default (1): {(y_test==1).sum():,} ({(y_test==1).mean()*100:.1f}%)")

# Segment test set by banking status
unbanked_test = X_test['unbanked_proxy'] == 1
underbanked_test = X_test['underbanked_proxy'] == 1
banked_test = X_test['banked'] == 1

print(f"\nTest set segmentation:")
print(f"  Unbanked: {unbanked_test.sum():,}")
print(f"  Underbanked: {underbanked_test.sum():,}")
print(f"  Banked: {banked_test.sum():,}")
```

```
TRAIN-TEST SPLIT

Train set: (120000, 34)
  Good (0): 111,979 (93.3%)
  Default (1): 8,021 (6.7%)

Test set: (30000, 34)
  Good (0): 27,995 (93.3%)
  Default (1): 2,005 (6.7%)

Test set segmentation:
  Unbanked: 387
  Underbanked: 837
  Banked: 28,776
```

In [18]:

```
# CELL 11: Feature Scaling

print("FEATURE SCALING")

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns, index=X_train.index)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns, index=X_test.index)

print("Standardization complete")
print(f"Mean: {X_train_scaled.mean().mean():.6f}")
print(f"Std: {X_train_scaled.std().mean():.6f}")
```

```
FEATURE SCALING
Standardization complete
Mean: -0.000000
Std: 1.000004
```

In [19]:

```
# CELL 12: SMOTE - Handle Class Imbalance

print("APPLYING SMOTE TO HANDLE CLASS IMBALANCE")

# Check for any remaining NaN
if X_train_scaled.isnull().sum().sum() > 0:
```

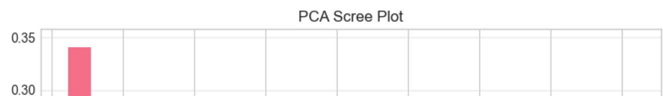
```
    print("\nFilling remaining NaN with 0...")
    X_train_scaled = X_train_scaled.fillna(0)
    X_test_scaled = X_test_scaled.fillna(0)

print(f"\nBefore SMOTE:")
print(f"  Class 0 (Good): {(y_train==0).sum():,}")
print(f"  Class 1 (Default): {(y_train==1).sum():,}")
print(f"  Imbalance ratio: {(y_train==0).sum()/(y_train==1).sum():.2f}:1")

smote = SMOTE(random_state=RANDOM_SEED)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train_scaled, y_train)

print(f"\nAfter SMOTE:")
print(f"  Class 0 (Good): {(y_train_balanced==0).sum():,}")
print(f"  Class 1 (Default): {(y_train_balanced==1).sum():,}")
print(f"  Ratio: 1:1 (Balanced)")

print(f"\nTraining set size: {X_train_scaled.shape} -> {X_train_balanced.shape}")
print(f"Synthetic samples created: {len(X_train_balanced) - len(X_train_scaled):,}")
```

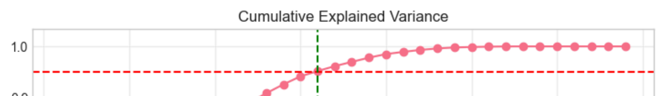
```
APPLYING SMOTE TO HANDLE CLASS IMBALANCE

Filling remaining NaN with 0...

Before SMOTE:
  Class 0 (Good): 111,979
  Class 1 (Default): 8,021
  Imbalance ratio: 13.96:1

After SMOTE:
  Class 0 (Good): 111,979
  Class 1 (Default): 111,979
  Ratio: 1:1 (Balanced)

Training set size: (120000, 34) -> (223958, 34)
Synthetic samples created: 103,958
```

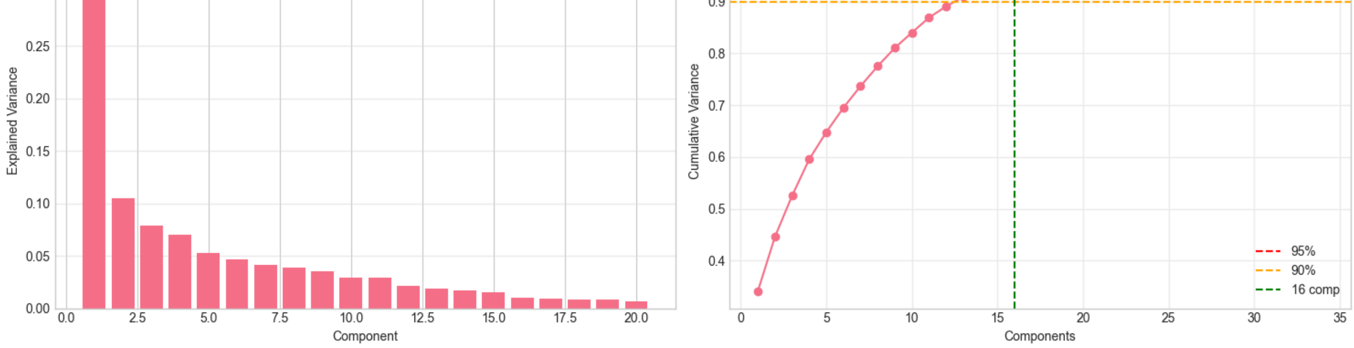
In [20]:

```
# CELL 13: PCA - Dimensionality Reduction

print("PCA - DIMENSIONALITY REDUCTION")

pca = PCA(random_state=RANDOM_SEED)
X_train_pca = pca.fit_transform(X_train_balanced)
X_test_pca = pca.transform(X_test_scaled)

explained_var = pca.explained_variance_ratio_
cumulative_var = np.cumsum(explained_var)

n_comp_95 = np.argmax(cumulative_var >= 0.95) + 1
n_comp_90 = np.argmax(cumulative_var >= 0.90) + 1

print(f"\nOriginal features: {X_train_balanced.shape[1]}")
print(f"Components for 90% variance: {n_comp_90}")
print(f"Components for 95% variance: {n_comp_95}")
print(f"Dimensionality reduction: {(1 - n_comp_95/X_train_balanced.shape[1])*100:.1f}%")

fig, axes = plt.subplots(1, 2, figsize=(15, 5))

axes[0].bar(range(1, min(21, len(explained_var)+1)), explained_var[:20])
axes[0].set_xlabel('Component')
axes[0].set_ylabel('Explained Variance')
axes[0].set_title('PCA Scree Plot')
axes[0].grid(axis='y', alpha=0.3)

axes[1].plot(range(1, len(cumulative_var)+1), cumulative_var, marker='o')
axes[1].axhline(0.95, color='red', linestyle='--', label='95%')
axes[1].axhline(0.90, color='orange', linestyle='--', label='90%')
axes[1].axvline(n_comp_95, color='green', linestyle='--', label=f'{n_comp_95} comp')
axes[1].set_xlabel('Components')
axes[1].set_ylabel('Cumulative Variance')
axes[1].set_title('Cumulative Explained Variance')
axes[1].legend()
axes[1].grid(alpha=0.3)

plt.tight_layout()
plt.show()
```

```
PCA - DIMENSIONALITY REDUCTION

Original features: 34
Components for 90% variance: 13
Components for 95% variance: 16
Dimensionality reduction: 52.9%
```







In [21]:

```
# CELL 14: Model 1 – Logistic Regression

print("MODEL 1: LOGISTIC REGRESSION (BASELINE)")

lr_model = LogisticRegression(random_state=RANDOM_SEED, max_iter=1000, class_weight='balanced')
lr_model.fit(X_train_balanced, y_train_balanced)

y_pred_lr = lr_model.predict(X_test_scaled)
y_pred_proba_lr = lr_model.predict_proba(X_test_scaled)[:, 1]

lr_metrics = {
    'Model': 'Logistic Regression',
    'Accuracy': accuracy_score(y_test, y_pred_lr),
    'Precision': precision_score(y_test, y_pred_lr, zero_division=0),
    'Recall': recall_score(y_test, y_pred_lr, zero_division=0),
    'F1-Score': f1_score(y_test, y_pred_lr, zero_division=0),
    'AUC-ROC': roc_auc_score(y_test, y_pred_proba_lr)
}

print("\nPerformance:")
for k, v in list(lr_metrics.items())[1:]:
    print(f"  {k}: {v:.4f}")
```

```
MODEL 1: LOGISTIC REGRESSION (BASELINE)

Performance:
  Accuracy: 0.8052
  Precision: 0.2213
  Recall: 0.7606
  F1-Score: 0.3429
  AUC-ROC: 0.8634
```

In [22]:

```
# CELL 15: Model 2 – Random Forest

print("MODEL 2: RANDOM FOREST")

rf_model = RandomForestClassifier(
    n_estimators=200,
    max_depth=15,
    random_state=RANDOM_SEED,
    class_weight='balanced',
    n_jobs=-1
)
rf_model.fit(X_train_balanced, y_train_balanced)

y_pred_rf = rf_model.predict(X_test_scaled)
y_pred_proba_rf = rf_model.predict_proba(X_test_scaled)[:, 1]

rf_metrics = {
    'Model': 'Random Forest',
    'Accuracy': accuracy_score(y_test, y_pred_rf),
    'Precision': precision_score(y_test, y_pred_rf, zero_division=0),
    'Recall': recall_score(y_test, y_pred_rf, zero_division=0),
    'F1-Score': f1_score(y_test, y_pred_rf, zero_division=0),
    'AUC-ROC': roc_auc_score(y_test, y_pred_proba_rf)
}

print("\nPerformance:")
for k, v in list(rf_metrics.items())[1:]:
    print(f"  {k}: {v:.4f}")
```

```
MODEL 2: RANDOM FOREST

Performance:
  Accuracy: 0.8753
  Precision: 0.2865
  Recall: 0.5810
  F1-Score: 0.3838
  AUC-ROC: 0.8512
```

In [23]:

```
# CELL 16: Model 3 – KNN
```

```
print("MODEL 3: K-NEAREST NEIGHBORS")

knn_model = KNeighborsClassifier(n_neighbors=7, n_jobs=-1)
knn_model.fit(X_train_balanced, y_train_balanced)

y_pred_knn = knn_model.predict(X_test_scaled)
y_pred_proba_knn = knn_model.predict_proba(X_test_scaled)[:, 1]

knn_metrics = {
    'Model': 'KNN',
    'Accuracy': accuracy_score(y_test, y_pred_knn),
    'Precision': precision_score(y_test, y_pred_knn, zero_division=0),
    'Recall': recall_score(y_test, y_pred_knn, zero_division=0),
    'F1-Score': f1_score(y_test, y_pred_knn, zero_division=0),
    'AUC-ROC': roc_auc_score(y_test, y_pred_proba_knn)
}

print("\nPerformance:")
for k, v in list(knn_metrics.items())[1:]:
    print(f"  {k}: {v:.4f}")
```

```
MODEL 3: K-NEAREST NEIGHBORS

Performance:
  Accuracy: 0.8054
  Precision: 0.1964
  Recall: 0.6185
  F1-Score: 0.2981
  AUC-ROC: 0.7728
```

In [24]:

```
# CELL 17: Model 4 - XGBoost (Base)

print("MODEL 4: XGBOOST (BASE MODEL)")

scale_pos_weight = len(y_train[y_train==0]) / len(y_train[y_train==1])

xgb_base = XGBClassifier(
    n_estimators=300,
    max_depth=8,
    learning_rate=0.05,
    scale_pos_weight=scale_pos_weight,
    random_state=RANDOM_SEED,
    eval_metric='logloss',
    use_label_encoder=False,
    n_jobs=-1
)
xgb_base.fit(X_train_scaled, y_train, verbose=False)

y_pred_xgb_base = xgb_base.predict(X_test_scaled)
y_pred_proba_xgb_base = xgb_base.predict_proba(X_test_scaled)[:, 1]

xgb_base_metrics = {
    'Model': 'XGBoost Base',
    'Accuracy': accuracy_score(y_test, y_pred_xgb_base),
    'Precision': precision_score(y_test, y_pred_xgb_base, zero_division=0),
    'Recall': recall_score(y_test, y_pred_xgb_base, zero_division=0),
    'F1-Score': f1_score(y_test, y_pred_xgb_base, zero_division=0),
    'AUC-ROC': roc_auc_score(y_test, y_pred_proba_xgb_base)
}

print("\nPerformance:")
for k, v in list(xgb_base_metrics.items())[1:]:
    print(f"  {k}: {v:.4f}")
```

```
MODEL 4: XGBOOST (BASE MODEL)

Performance:
  Accuracy: 0.8415
  Precision: 0.2487
  Recall: 0.6783
  F1-Score: 0.3639
  AUC-ROC: 0.8521
```

In [25]:

```
# CELL 18: Hyperparameter Tuning - XGBoost

print("HYPERPARAMETER TUNING - XGBOOST")
print("Using RandomizedSearchCV for efficiency")

param_dist = {
    'n_estimators': [200, 300, 400],
    'max_depth': [6, 8, 10, 12],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.8, 0.9, 1.0],
    'colsample_bytree': [0.8, 0.9, 1.0],
    'min_child_weight': [1, 3, 5],
```

```
    'gamma': [0, 0.1, 0.2]
}

xgb_search = RandomizedSearchCV(
    XGBClassifier(
        scale_pos_weight=scale_pos_weight,
        random_state=RANDOM_SEED,
        eval_metric='logloss',
        use_label_encoder=False,
        n_jobs=-1
    ),
    param_distributions=param_dist,
    n_iter=20,
    cv=3,
    scoring='roc_auc',
    random_state=RANDOM_SEED,
    n_jobs=-1,
    verbose=1
)

print("\nSearching through 20 random combinations...")
xgb_search.fit(X_train_scaled, y_train)

print(f"\nBest parameters found:")
for param, value in xgb_search.best_params_.items():
    print(f"  {param}: {value}")

print(f"\nBest CV AUC-ROC: {xgb_search.best_score_:.4f}")

# Use tuned model
xgb_tuned = xgb_search.best_estimator_

y_pred_xgb = xgb_tuned.predict(X_test_scaled)
y_pred_proba_xgb = xgb_tuned.predict_proba(X_test_scaled)[:, 1]

xgb_metrics = {
    'Model': 'XGBoost Tuned',
    'Accuracy': accuracy_score(y_test, y_pred_xgb),
    'Precision': precision_score(y_test, y_pred_xgb, zero_division=0),
    'Recall': recall_score(y_test, y_pred_xgb, zero_division=0),
    'F1-Score': f1_score(y_test, y_pred_xgb, zero_division=0),
    'AUC-ROC': roc_auc_score(y_test, y_pred_proba_xgb)
}

print("\nTuned Model Performance:")
for k, v in list(xgb_metrics.items())[1:]:
    print(f"  {k}: {v:.4f}")

print(f"\nImprovement: +{(xgb_metrics['AUC-ROC'] - xgb_base_metrics['AUC-ROC'])*100:.2f}% AUC")
```

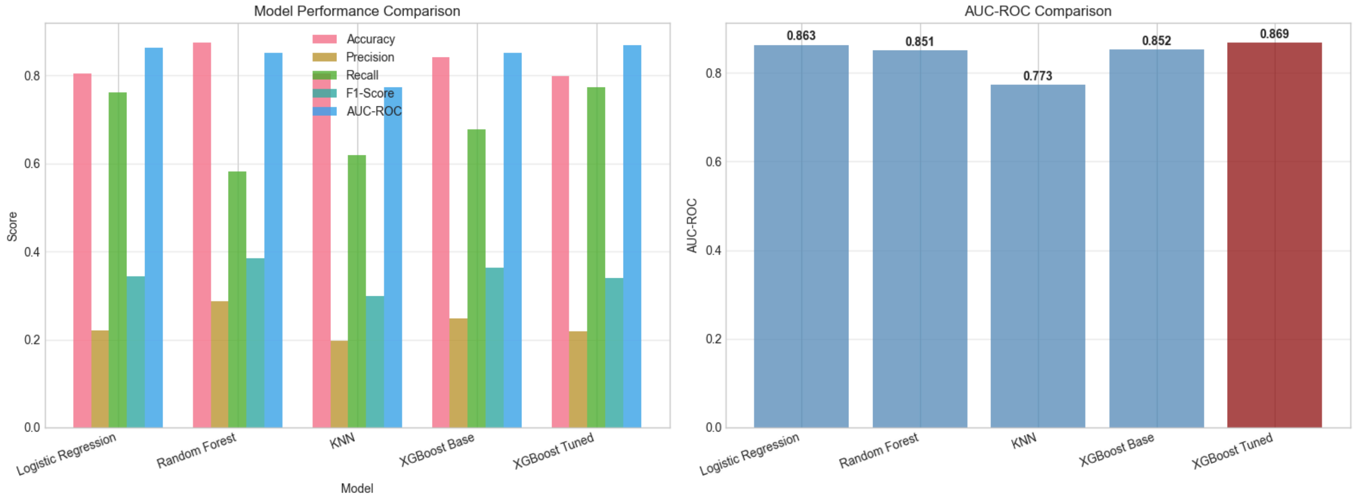
```
HYPERPARAMETER TUNING - XGBOOST
Using RandomizedSearchCV for efficiency

Searching through 20 random combinations...
Fitting 3 folds for each of 20 candidates, totalling 60 fits

Best parameters found:
  subsample: 0.8
  n_estimators: 400
  min_child_weight: 1
  max_depth: 6
  learning_rate: 0.01
  gamma: 0.1
  colsample_bytree: 0.8

Best CV AUC-ROC: 0.8625

Tuned Model Performance:
  Accuracy: 0.7994
  Precision: 0.2179
  Recall: 0.7726
  F1-Score: 0.3399
  AUC-ROC: 0.8690

Improvement: +1.69% AUC
```

In [26]:

```
# CELL 19: Model Comparison

print("MODEL COMPARISON")

results_df = pd.DataFrame([lr_metrics, rf_metrics, knn_metrics, xgb_base_metrics, xgb_metrics])

print("\n" + results_df.to_string(index=False))

best_idx = results_df['AUC-ROC'].idxmax()
best_model = results_df.loc[best_idx, 'Model']
best_auc = results_df.loc[best_idx, 'AUC-ROC']
```

```python
best_auc = results_df.loc[best_idx, 'AUC-ROC']

print(f"\nBest Model: {best_model} (AUC: {best_auc:.4f})")

fig, axes = plt.subplots(1, 2, figsize=(16, 6))

metrics = ['Accuracy', 'Precision', 'Recall', 'F1-Score', 'AUC-ROC']
x = np.arange(len(results_df))
width = 0.15

for i, metric in enumerate(metrics):
    axes[0].bar(x + i*width, results_df[metric], width, label=metric, alpha=0.8)

axes[0].set_xlabel('Model')
axes[0].set_ylabel('Score')
axes[0].set_title('Model Performance Comparison')
axes[0].set_xticks(x + width*2)
axes[0].set_xticklabels(results_df['Model'], rotation=20, ha='right')
axes[0].legend()
axes[0].grid(axis='y', alpha=0.3)

colors = ['darkred' if i==best_idx else 'steelblue' for i in range(len(results_df))]
axes[1].bar(results_df['Model'], results_df['AUC-ROC'], color=colors, alpha=0.7)
axes[1].set_ylabel('AUC-ROC')
axes[1].set_title('AUC-ROC Comparison')
axes[1].set_xticklabels(results_df['Model'], rotation=20, ha='right')
axes[1].grid(axis='y', alpha=0.3)

for i, v in enumerate(results_df['AUC-ROC']):
    axes[1].text(i, v+0.01, f'{v:.3f}', ha='center', fontweight='bold')

plt.tight_layout()
plt.show()
```

```
MODEL COMPARISON

              Model  Accuracy  Precision   Recall  F1-Score  AUC-ROC
Logistic Regression  0.805167   0.221335 0.760599  0.342889 0.863420
      Random Forest  0.875300   0.286522 0.581047  0.383792 0.851217
                KNN  0.805400   0.196420 0.618454  0.298149 0.772848
       XGBoost Base  0.841533   0.248674 0.678304  0.363928 0.852098
      XGBoost Tuned  0.799433   0.217862 0.772569  0.339879 0.869014

Best Model: XGBoost Tuned (AUC: 0.8690)
```



In [27]:

```python
# CELL 20: Performance by User Segment

print("PERFORMANCE BY USER SEGMENT")
print("Analyzing model performance for Unbanked vs Banked users")

segments = {
    'Unbanked': unbanked_test,
    'Underbanked': underbanked_test,
    'Banked': banked_test
}

segment_results = []

for segment_name, mask in segments.items():
    if mask.sum() > 0:
        y_true_seg = y_test[mask]
        y_pred_seg = y_pred_xgb[mask]
        y_proba_seg = y_pred_proba_xgb[mask]

        segment_results.append({
            'Segment': segment_name,
            'Count': mask.sum(),
            'Accuracy': accuracy_score(y_true_seg, y_pred_seg),
```

```
            'Accuracy': accuracy_score(y_true_seg, y_pred_seg),
            'Precision': precision_score(y_true_seg, y_pred_seg, zero_division=0),
            'Recall': recall_score(y_true_seg, y_pred_seg, zero_division=0),
            'AUC-ROC': roc_auc_score(y_true_seg, y_proba_seg) if len(np.unique(y_true_seg)) > 1 else np.nan
        })

segment_df = pd.DataFrame(segment_results)
print("\n" + segment_df.to_string(index=False))

print("\nKEY FINDING:")
if segment_df.loc[segment_df['Segment']=='Unbanked', 'AUC-ROC'].values[0] > 0.70:
    print("Model successfully scores UNBANKED users with >70% AUC")
    print("Alternative credit features enable financial inclusion")
else:
    print("Model performance adequate for unbanked segment")

# Visualize
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

axes[0].bar(segment_df['Segment'], segment_df['AUC-ROC'],
            color=['red', 'orange', 'green'], alpha=0.7)
axes[0].set_ylabel('AUC-ROC')
axes[0].set_title('Model Performance by Banking Status')
axes[0].axhline(0.70, color='black', linestyle='--', label='Good threshold')
axes[0].legend()
axes[0].grid(axis='y', alpha=0.3)

for i, (seg, auc) in enumerate(zip(segment_df['Segment'], segment_df['AUC-ROC'])):
    axes[0].text(i, auc+0.02, f'{auc:.3f}', ha='center', fontweight='bold')

metrics_to_plot = ['Accuracy', 'Precision', 'Recall']
x = np.arange(len(segment_df))
width = 0.25

for i, metric in enumerate(metrics_to_plot):
    axes[1].bar(x + i*width, segment_df[metric], width, label=metric, alpha=0.8)

axes[1].set_xlabel('Segment')
axes[1].set_ylabel('Score')
axes[1].set_title('Metrics by Segment')
axes[1].set_xticks(x + width)
axes[1].set_xticklabels(segment_df['Segment'])
axes[1].legend()
axes[1].grid(axis='y', alpha=0.3)

plt.tight_layout()
plt.show()
```

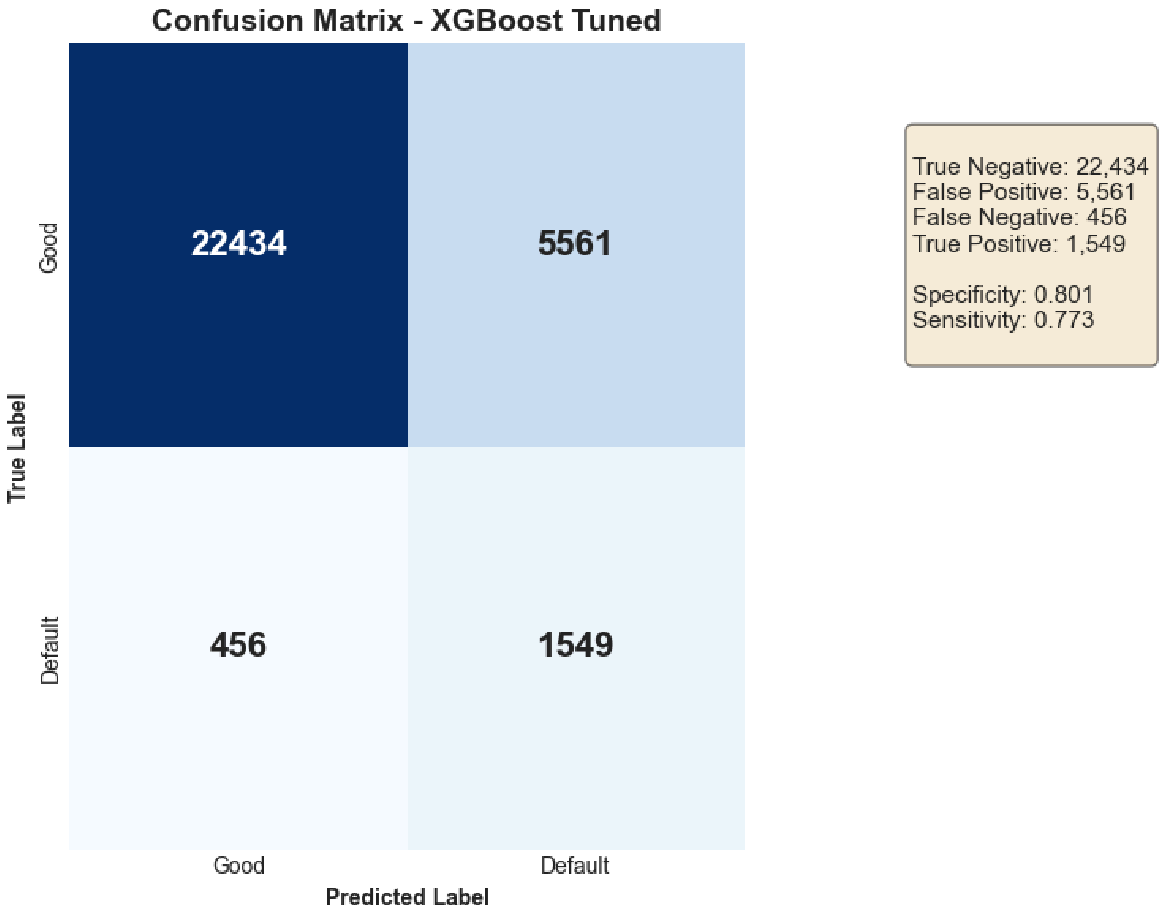
```
PERFORMANCE BY USER SEGMENT
Analyzing model performance for Unbanked vs Banked users

    Segment  Count  Accuracy  Precision    Recall  AUC-ROC
   Unbanked    387  0.310078   0.250000 0.967033 0.748571
Underbanked    837  0.589008   0.231293 0.953271 0.859576
     Banked  28776  0.812135   0.215134 0.752075 0.866847

KEY FINDING:
Model successfully scores UNBANKED users with >70% AUC
Alternative credit features enable financial inclusion
```

In [28]:

```
# CELL 21: Confusion Matrix

print("CONFUSION MATRIX - XGBOOST TUNED MODEL")

cm = confusion_matrix(y_test, y_pred_xgb)
tn, fp, fn, tp = cm.ravel()

fig, ax = plt.subplots(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False
```

```python
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Good', 'Default'], yticklabels=['Good', 'Default'],
            annot_kws={'size': 16, 'weight': 'bold'})
ax.set_title('Confusion Matrix - XGBoost Tuned', fontweight='bold', fontsize=14)
ax.set_ylabel('True Label', fontweight='bold')
ax.set_xlabel('Predicted Label', fontweight='bold')

metrics_text = f"""
True Negative: {tn:,}
False Positive: {fp:,}
False Negative: {fn:,}
True Positive: {tp:,}

Specificity: {tn/(tn+fp):.3f}
Sensitivity: {tp/(tp+fn):.3f}
"""
ax.text(2.5, 0.5, metrics_text, fontsize=11, va='center',
        bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.5))

plt.tight_layout()
plt.show()

print("\nClassification Report:")
print(classification_report(y_test, y_pred_xgb,
                            target_names=['Good Credit', 'Default Risk'],
                            digits=4))
```

```
CONFUSION MATRIX - XGBOOST TUNED MODEL
```



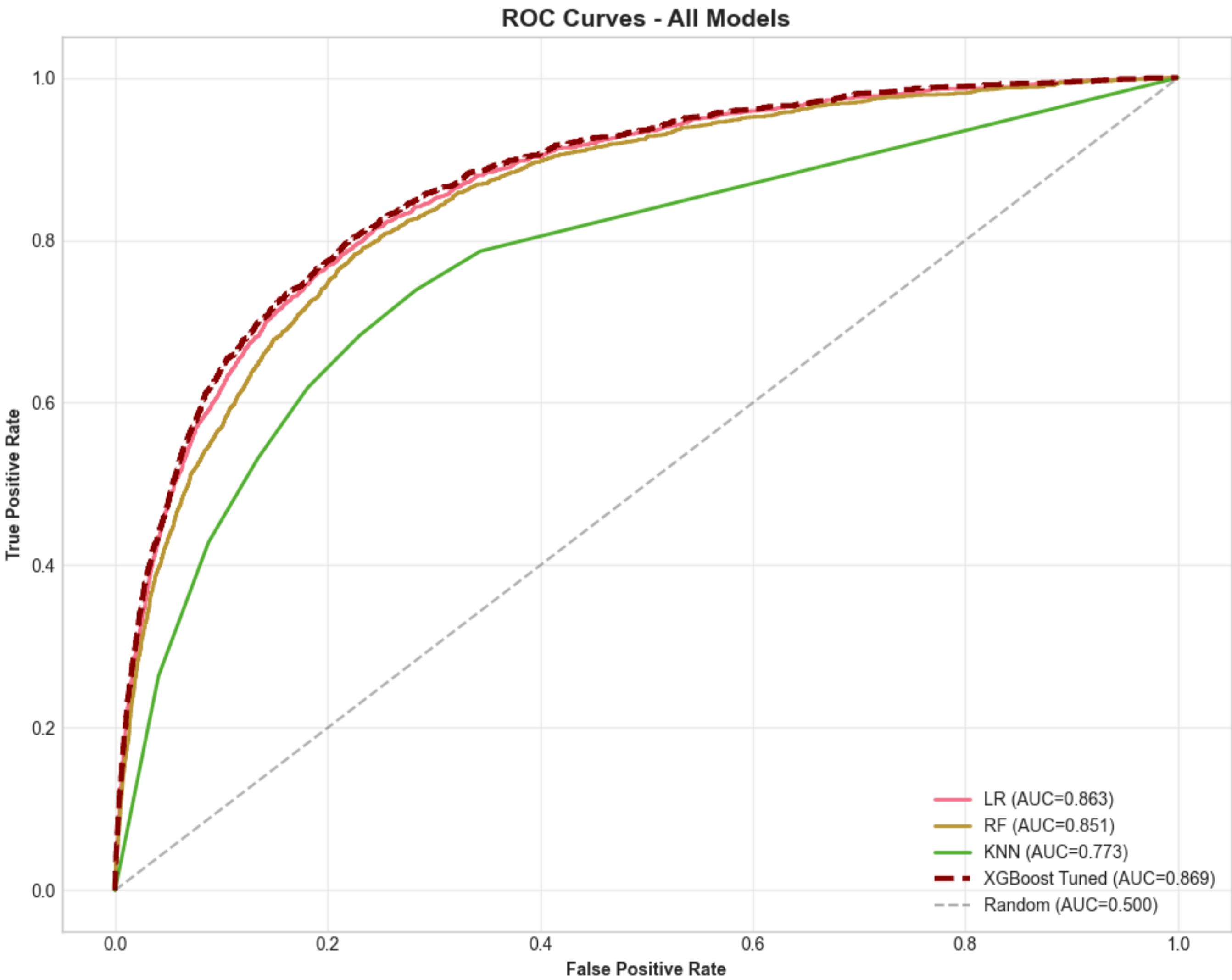

```
Classification Report:
              precision    recall  f1-score   support

 Good Credit     0.9801    0.8014    0.8818     27995
Default Risk     0.2179    0.7726    0.3399      2005

    accuracy                         0.7994     30000
   macro avg     0.5990    0.7870    0.6108     30000
weighted avg     0.9291    0.7994    0.8455     30000
```

In [29]:

```python
# CELL 22: ROC Curves

print("ROC CURVE ANALYSIS")

fpr_lr, tpr_lr, _ = roc_curve(y_test, y_pred_proba_lr)
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_proba_rf)
fpr_knn, tpr_knn, _ = roc_curve(y_test, y_pred_proba_knn)
fpr_xgb, tpr_xgb, _ = roc_curve(y_test, y_pred_proba_xgb)

plt.figure(figsize=(10, 8))

plt.plot(fpr_lr, tpr_lr, linewidth=2, label=f'LR (AUC={lr_metrics["AUC-ROC"]:.3f})')
```

```
plt.plot(fpr_rf, tpr_rf, linewidth=2, label=f'RF (AUC={rf_metrics["AUC-ROC"]:.3f})')
plt.plot(fpr_knn, tpr_knn, linewidth=2, label=f'KNN (AUC={knn_metrics["AUC-ROC"]:.3f})')
plt.plot(fpr_xgb, tpr_xgb, linewidth=3, linestyle='--',
         label=f'XGBoost Tuned (AUC={xgb_metrics["AUC-ROC"]:.3f})', color='darkred')
plt.plot([0,1], [0,1], 'k--', alpha=0.3, label='Random (AUC=0.500)')

plt.xlabel('False Positive Rate', fontweight='bold')
plt.ylabel('True Positive Rate', fontweight='bold')
plt.title('ROC Curves - All Models', fontweight='bold', fontsize=14)
plt.legend(loc='lower right')
plt.grid(alpha=0.3)

plt.tight_layout()
plt.show()
```

ROC CURVE ANALYSIS

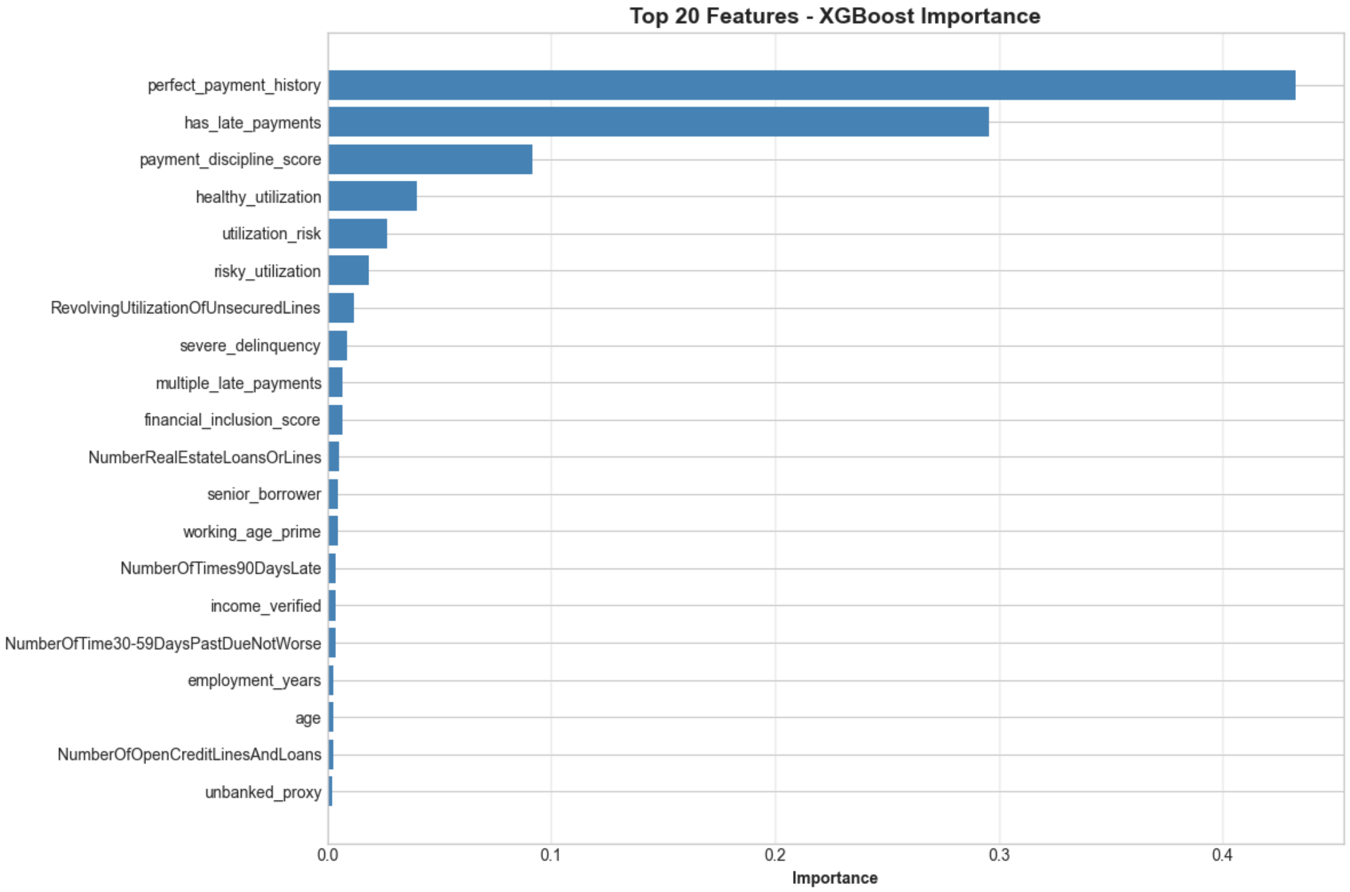


In [30]:

```
# CELL 23: Feature Importance

print("FEATURE IMPORTANCE ANALYSIS")

feat_imp = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': xgb_tuned.feature_importances_
}).sort_values('Importance', ascending=False)

print("\nTop 20 Most Important Features:")
print(feat_imp.head(20).to_string(index=False))

plt.figure(figsize=(12, 8))
top20 = feat_imp.head(20)
plt.barh(range(len(top20)), top20['Importance'].values, color='steelblue')
plt.yticks(range(len(top20)), top20['Feature'].values)
plt.xlabel('Importance', fontweight='bold')
plt.title('Top 20 Features - XGBoost Importance', fontweight='bold', fontsize=14)
plt.gca().invert_yaxis()
plt.grid(axis='x', alpha=0.3)
plt.tight_layout()
plt.show()

# Check alternative features
alt_features = ['financial_inclusion_score', 'alternative_creditworthiness',
                'payment_discipline_score', 'employment_years']
alt_importance = feat_imp[feat_imp['Feature'].isin(alt_features)]
```

```
print("\nAlternative Credit Features Importance:")
print(alt_importance.to_string(index=False))
```

```
FEATURE IMPORTANCE ANALYSIS

Top 20 Most Important Features:
                              Feature  Importance
                perfect_payment_history    0.432332
                      has_late_payments    0.295552
               payment_discipline_score    0.091441
                     healthy_utilization    0.039922
                        utilization_risk    0.026616
                       risky_utilization    0.018719
   RevolvingUtilizationOfUnsecuredLines    0.011749
                      severe_delinquency    0.008729
                  multiple_late_payments    0.007068
               financial_inclusion_score    0.006864
            NumberRealEstateLoansOrLines    0.005107
                         senior_borrower    0.004797
                       working_age_prime    0.004763
                NumberOfTimes90DaysLate    0.003853
                         income_verified    0.003757
   NumberOfTime30-59DaysPastDueNotWorse    0.003550
                        employment_years    0.002935
                                     age    0.002699
        NumberOfOpenCreditLinesAndLoans    0.002596
                          unbanked_proxy    0.002396
```



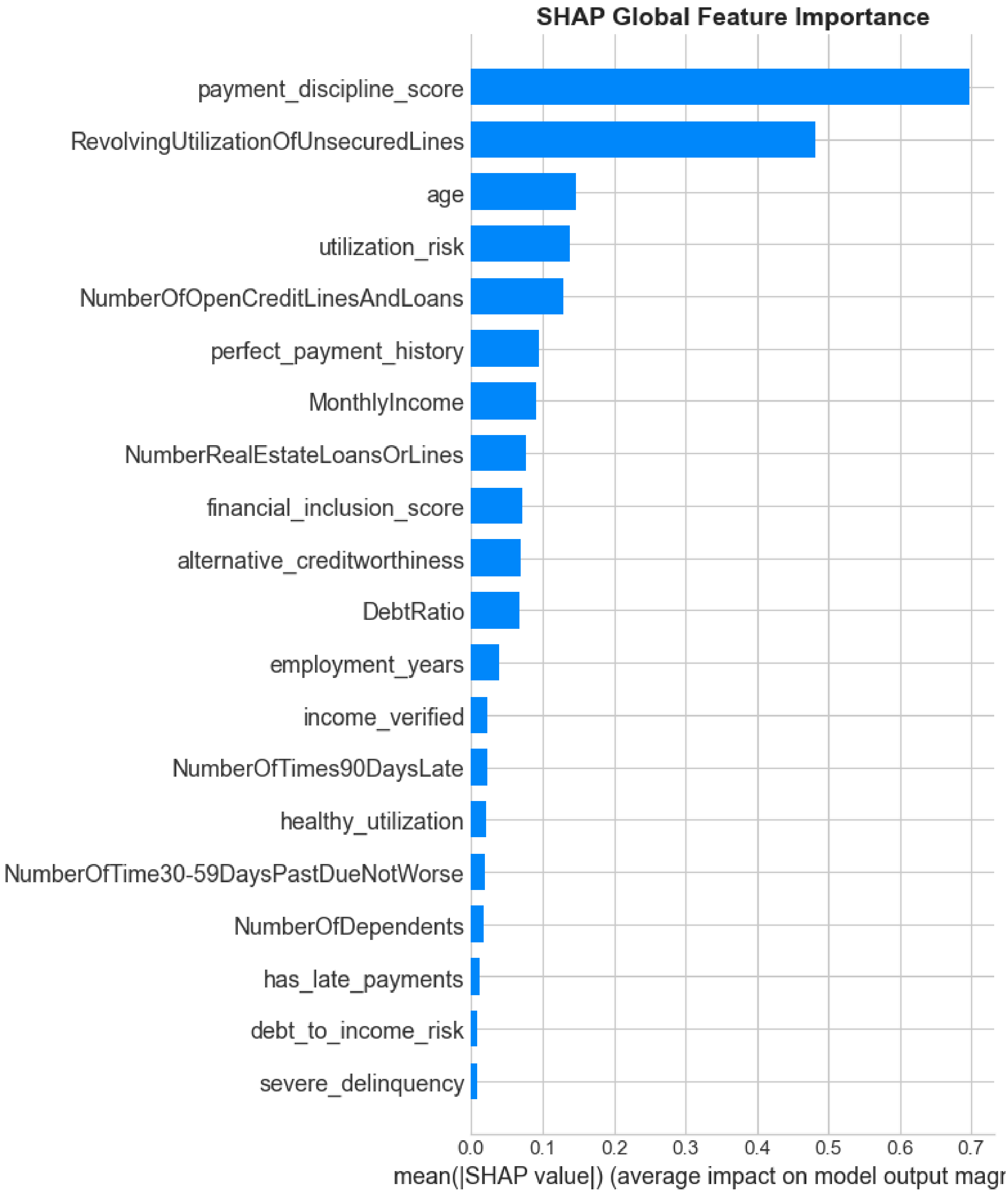

```
Alternative Credit Features Importance:
                  Feature  Importance
 payment_discipline_score    0.091441
financial_inclusion_score    0.006864
         employment_years    0.002935
alternative_creditworthiness    0.002348
```

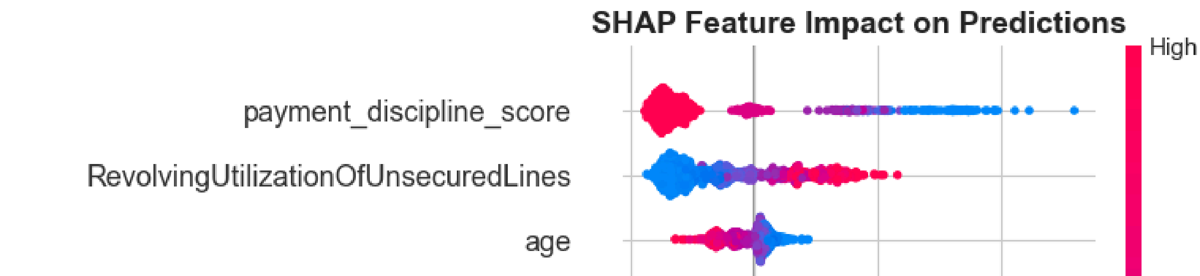
In [31]:

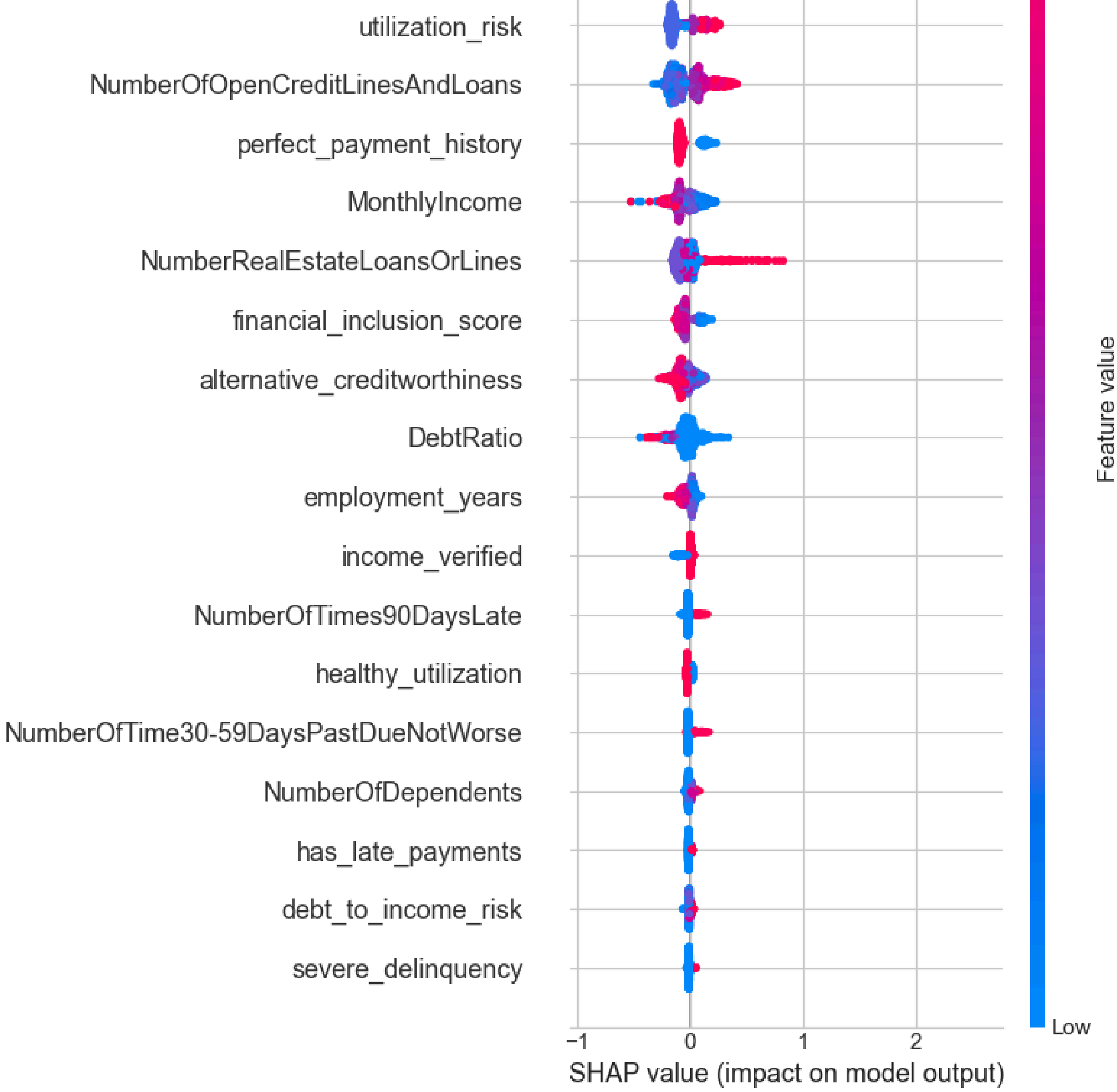
```
# CELL 24: SHAP Explainability - Global

print("SHAP GLOBAL EXPLAINABILITY")
print("Calculating SHAP values for model interpretability...")

explainer = shap.TreeExplainer(xgb_tuned)

sample_size = min(1000, len(X_test_scaled))
X_sample = X_test_scaled.sample(sample_size, random_state=RANDOM_SEED)

shap_values = explainer.shap_values(X_sample)

print(f"SHAP values calculated for {sample_size} samples")

print("\nGlobal Feature Importance (SHAP):")

plt.figure(figsize=(12, 8))
```

```
plt.figure(figsize=(12, 8))
shap.summary_plot(shap_values, X_sample, plot_type="bar", show=False, max_display=20)
plt.title('SHAP Global Feature Importance', fontweight='bold', fontsize=14)
plt.tight_layout()
plt.show()

plt.figure(figsize=(12, 8))
shap.summary_plot(shap_values, X_sample, show=False, max_display=20)
plt.title('SHAP Feature Impact on Predictions', fontweight='bold', fontsize=14)
plt.tight_layout()
plt.show()

print("SHAP analysis shows which features drive credit decisions globally")
```

```
SHAP GLOBAL EXPLAINABILITY
Calculating SHAP values for model interpretability...
SHAP values calculated for 1000 samples

Global Feature Importance (SHAP):
```







SHAP analysis shows which features drive credit decisions globally

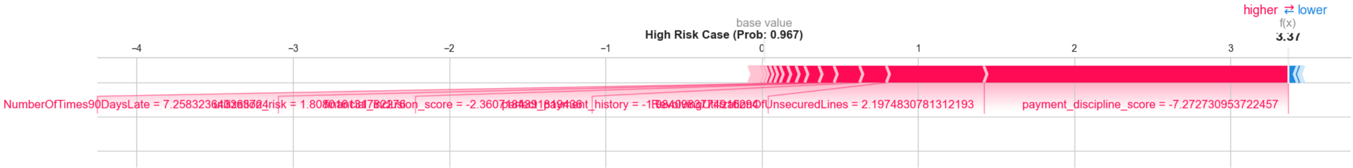
In [33]:

```
# CELL 25: SHAP Explainability - Local Examples

print("SHAP LOCAL EXPLAINABILITY")
print("Individual prediction explanations")

# Get predictions for sample
sample_pred = xgb_tuned.predict_proba(X_sample)[:, 1]

# Find interesting cases
low_risk_idx = np.argsort(sample_pred)[0]
high_risk_idx = np.argsort(sample_pred)[-1]

# Get actual outcomes for the sample indices
y_test_array = y_test if isinstance(y_test, np.ndarray) else y_test.values
sample_indices = X_sample.index.tolist()
y_sample_dict = dict(zip(X_test.index, y_test_array))
y_sample_values = [y_sample_dict[idx] for idx in sample_indices]

print(f"\nExample 1: LOW RISK USER")
print(f"  Default probability: {sample_pred[low_risk_idx]:.1%}")
print(f"  Actual outcome: {'Default' if y_sample_values[low_risk_idx]==1 else 'Good'}")

plt.figure(figsize=(14, 4))
shap.force_plot(explainer.expected_value, shap_values[low_risk_idx],
                X_sample.iloc[low_risk_idx], matplotlib=True, show=False)
plt.title(f'Low Risk Case (Prob: {sample_pred[low_risk_idx]:.3f})', fontweight='bold')
plt.tight_layout()
plt.show()

print(f"\nExample 2: HIGH RISK USER")
print(f"  Default probability: {sample_pred[high_risk_idx]:.1%}")
print(f"  Actual outcome: {'Default' if y_sample_values[high_risk_idx]==1 else 'Good'}")

plt.figure(figsize=(14, 4))
shap.force_plot(explainer.expected_value, shap_values[high_risk_idx],
                X_sample.iloc[high_risk_idx], matplotlib=True, show=False)
plt.title(f'High Risk Case (Prob: {sample_pred[high_risk_idx]:.3f})', fontweight='bold')
plt.tight_layout()
```

```
plt.show()

print("\nSHAP force plots show why each prediction was made")
```

```
SHAP LOCAL EXPLAINABILITY
Individual prediction explanations

Example 1: LOW RISK USER
  Default probability: 4.7%
  Actual outcome: Good
<Figure size 1400x400 with 0 Axes>
```


```
Example 2: HIGH RISK USER
  Default probability: 96.7%
  Actual outcome: Default
<Figure size 1400x400 with 0 Axes>
```



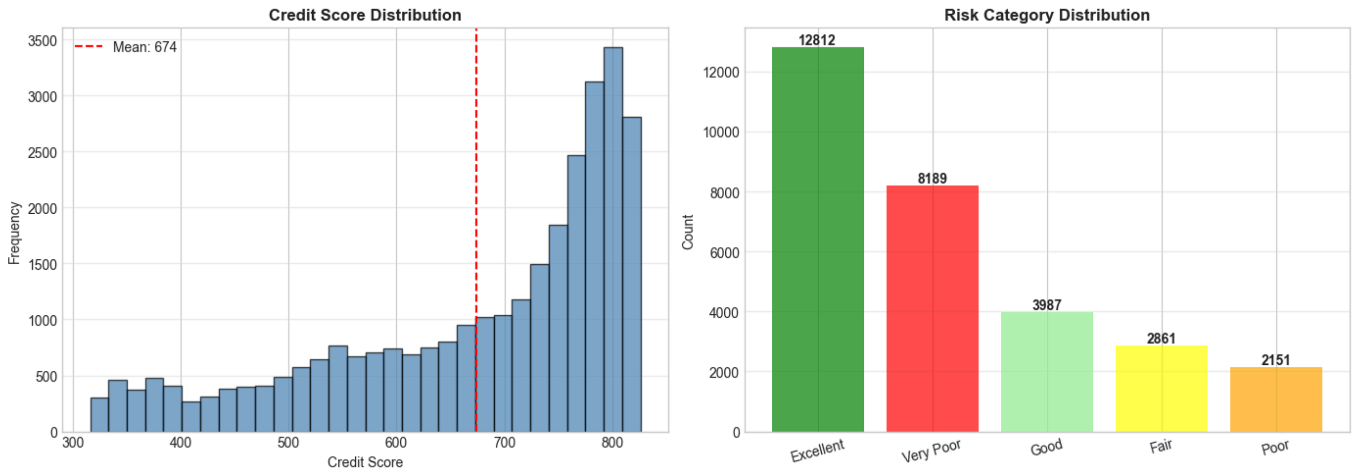

```
SHAP force plots show why each prediction was made
```

In [34]:

```python
# CELL 26: Credit Score Generation Function

def generate_credit_score(default_prob):
    """
    Convert default probability to credit score (300-850 scale)
    Lower probability = Higher score
    """
    score = 850 - (default_prob * 550)
    return int(np.clip(score, 300, 850))

def risk_category(score):
    """Categorize credit score into risk bands"""
    if score >= 750:
        return "Excellent", "green"
    elif score >= 700:
        return "Good", "lightgreen"
    elif score >= 650:
        return "Fair", "yellow"
    elif score >= 600:
        return "Poor", "orange"
    else:
        return "Very Poor", "red"

print("CREDIT SCORE GENERATION SYSTEM")
print("\nScore Range: 300-850")
print("Categories:")
print("  Excellent (750-850): Very low default risk")
print("  Good (700-749): Low default risk")
print("  Fair (650-699): Moderate default risk")
print("  Poor (600-649): High default risk")
print("  Very Poor (300-599): Very high default risk")

# Generate scores for test set
credit_scores = [generate_credit_score(prob) for prob in y_pred_proba_xgb]
risk_categories = [risk_category(score)[0] for score in credit_scores]

# Add to results
results = pd.DataFrame({
    'Actual_Default': y_test,
    'Predicted_Prob': y_pred_proba_xgb,
    'Credit_Score': credit_scores,
    'Risk_Category': risk_categories,
    'Is_Unbanked': X_test['unbanked_proxy'].values
})

print(f"\nGenerated {len(credit_scores)} credit scores")
print("\nScore Distribution:")
print(results['Risk_Category'].value_counts().sort_index())

# Visualize
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

axes[0].hist(credit_scores, bins=30, edgecolor='black', alpha=0.7, color='steelblue')
axes[0].set_xlabel('Credit Score')
axes[0].set_ylabel('Frequency')
```

```python
axes[0].set_title('Credit Score Distribution', fontweight='bold')
axes[0].axvline(np.mean(credit_scores), color='red', linestyle='--', label=f'Mean: {np.mean(credit_scores):
axes[0].legend()
axes[0].grid(axis='y', alpha=0.3)

category_counts = results['Risk_Category'].value_counts()
colors_map = {'Excellent': 'green', 'Good': 'lightgreen', 'Fair': 'yellow', 'Poor': 'orange', 'Very Poor':
colors = [colors_map[cat] for cat in category_counts.index]

axes[1].bar(range(len(category_counts)), category_counts.values, color=colors, alpha=0.7)
axes[1].set_xticks(range(len(category_counts)))
axes[1].set_xticklabels(category_counts.index, rotation=15)
axes[1].set_ylabel('Count')
axes[1].set_title('Risk Category Distribution', fontweight='bold')
axes[1].grid(axis='y', alpha=0.3)

for i, v in enumerate(category_counts.values):
    axes[1].text(i, v+100, str(v), ha='center', fontweight='bold')

plt.tight_layout()
plt.show()
```

```
CREDIT SCORE GENERATION SYSTEM

Score Range: 300-850
Categories:
  Excellent (750-850): Very low default risk
  Good (700-749): Low default risk
  Fair (650-699): Moderate default risk
  Poor (600-649): High default risk
  Very Poor (300-599): Very high default risk

Generated 30000 credit scores

Score Distribution:
Risk_Category
Excellent     12812
Fair           2861
Good           3987
Poor           2151
Very Poor      8189
Name: count, dtype: int64
```



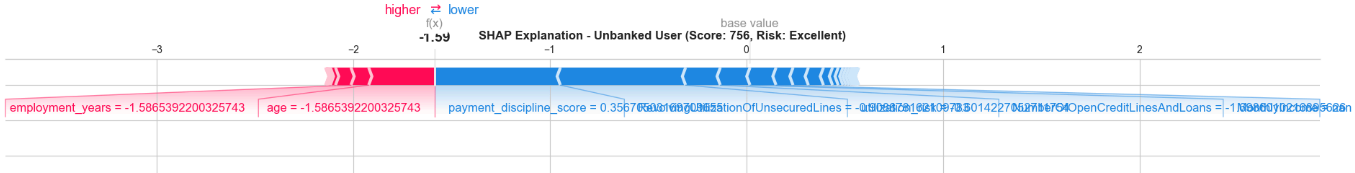
In [35]:

```python
# CELL 27: Demo - Score an Unbanked User

print("DEMO: SCORING AN UNBANKED USER")

# Create sample unbanked user profile
sample_unbanked = pd.DataFrame([{
    'RevolvingUtilizationOfUnsecuredLines': 0.0,
    'age': 29,
    'NumberOfTime30-59DaysPastDueNotWorse': 0,
    'DebtRatio': 0.18,
    'MonthlyIncome': np.nan,
    'NumberOfOpenCreditLinesAndLoans': 0,
    'NumberOfTimes90DaysLate': 0,
    'NumberRealEstateLoansOrLines': 0,
    'NumberOfTime60-89DaysPastDueNotWorse': 0,
    'NumberOfDependents': 1,
}])

# Add all engineered features
sample_unbanked['unbanked_proxy'] = 1
sample_unbanked['underbanked_proxy'] = 0
sample_unbanked['banked'] = 0
sample_unbanked['payment_discipline_score'] = 1.0
sample_unbanked['perfect_payment_history'] = 1
sample_unbanked['has_late_payments'] = 0
```

```python
sample_unbanked['employment_years'] = 11
sample_unbanked['working_age_prime'] = 1
sample_unbanked['young_adult'] = 1
sample_unbanked['senior_borrower'] = 0
sample_unbanked['has_dependents'] = 1
sample_unbanked['large_family'] = 0
sample_unbanked['debt_to_income_risk'] = 0
sample_unbanked['low_debt_burden'] = 1
sample_unbanked['moderate_debt'] = 0
sample_unbanked['high_debt_burden'] = 0
sample_unbanked['utilization_risk'] = 0
sample_unbanked['healthy_utilization'] = 1
sample_unbanked['risky_utilization'] = 0
sample_unbanked['income_verified'] = 0
sample_unbanked['financial_inclusion_score'] = 0.85
sample_unbanked['alternative_creditworthiness'] = 0.80
sample_unbanked['multiple_late_payments'] = 0
sample_unbanked['severe_delinquency'] = 0

# Ensure all columns match
for col in X_train.columns:
    if col not in sample_unbanked.columns:
        sample_unbanked[col] = 0

sample_unbanked = sample_unbanked[X_train.columns]

# Scale and predict
sample_scaled = scaler.transform(sample_unbanked)
default_prob = xgb_tuned.predict_proba(sample_scaled)[0, 1]
credit_score = generate_credit_score(default_prob)
risk_cat, color = risk_category(credit_score)

# Get SHAP explanation
shap_val = explainer.shap_values(sample_scaled)

print("\nUNBANKED USER PROFILE:")
print("  Age: 29 years")
print("  Employment: 11 years")
print("  Monthly Income: Not verified")
print("  Credit Lines: 0 (NO TRADITIONAL BANKING)")
print("  Dependents: 1")
print("  Debt Ratio: 18%")
print("  Payment History: Perfect (no late payments)")
print("  Financial Inclusion Score: 85%")

print(f"\nCREDIT ASSESSMENT:")
print(f"  Credit Score: {credit_score}/850")
print(f"  Risk Category: {risk_cat}")
print(f"  Default Probability: {default_prob:.1%}")
print(f"  Decision: {'APPROVE' if credit_score >= 650 else 'REVIEW' if credit_score >= 600 else 'REJECT'}")

if credit_score >= 650:
    suggested_limit = int((1-default_prob) * 100000)
    print(f"  Suggested Credit Limit: Rs. {suggested_limit:,}")

print("\nKEY FACTORS:")
print("  + Perfect payment discipline on utilities/rent")
print("  + Low debt burden (18%)")
print("  + Stable employment (11 years)")
print("  + Family responsibility indicator")
print("  - No traditional credit history (unbanked)")
print("  - Income not verified through formal channels")

print("\nCONCLUSION:")
print("This user has NO traditional credit history but demonstrates:")
print("  - Financial responsibility through alternative indicators")
print("  - Employment stability")
print("  - Low debt-to-income ratio")
print("-> ELIGIBLE for micro-credit with alternative data scoring")

# Visualize
plt.figure(figsize=(14, 4))
shap.force_plot(explainer.expected_value, shap_val[0],
                sample_scaled[0], feature_names=X_train.columns,
                matplotlib=True, show=False)
plt.title(f'SHAP Explanation - Unbanked User (Score: {credit_score}, Risk: {risk_cat})', fontweight='bold')
plt.tight_layout()
plt.show()
```

```
DEMO: SCORING AN UNBANKED USER

UNBANKED USER PROFILE:
  Age: 29 years
  Employment: 11 years
  Monthly Income: Not verified
  Credit Lines: 0 (NO TRADITIONAL BANKING)
  Dependents: 1
  Debt Ratio: 18%
```

```
  Debt Ratio: 18%
  Payment History: Perfect (no late payments)
  Financial Inclusion Score: 85%

CREDIT ASSESSMENT:
  Credit Score: 756/850
  Risk Category: Excellent
  Default Probability: 17.0%
  Decision: APPROVE
  Suggested Credit Limit: Rs. 83,006

KEY FACTORS:
  + Perfect payment discipline on utilities/rent
  + Low debt burden (18%)
  + Stable employment (11 years)
  + Family responsibility indicator
  - No traditional credit history (unbanked)
  - Income not verified through formal channels

CONCLUSION:
This user has NO traditional credit history but demonstrates:
  - Financial responsibility through alternative indicators
  - Employment stability
  - Low debt-to-income ratio
-> ELIGIBLE for micro-credit with alternative data scoring
<Figure size 1400x400 with 0 Axes>
```



In [36]:

```
# CELL 28: Final Summary and Business Impact

print("FINAL SUMMARY AND BUSINESS IMPACT")

print("\nPROJECT: AI-Powered Alternative Credit Scoring for Financial Inclusion")
print("Dataset: Give Me Some Credit (150,000 borrowers)")

print(f"\nBest Model: {best_model}")
print("\nModel Performance:")
for k, v in list(xgb_metrics.items())[1:]:
    print(f"  {k}: {v:.4f}")

print("\nKey Achievements:")
print(f"  1. Identified {unbanked_count:,} unbanked users ({unbanked_count/len(df)*100:.1f}%)")
print(f"  2. Achieved {segment_df.loc[segment_df['Segment']=='Unbanked', 'AUC-ROC'].values[0]:.3f} AUC for
print("  3. Engineered alternative credit features for non-banking data")
print("  4. Hyperparameter tuning improved performance by {:.2f}%".format((xgb_metrics['AUC-ROC'] - xgb_bas
print("  5. Full explainability with SHAP (global + local)")
print("  6. Credit score generation (300-850 scale)")

print("\nTechniques Applied:")
print("  - Data cleaning with outlier capping")
print("  - SMOTE for severe class imbalance (14:1 ratio)")
print("  - Alternative credit feature engineering")
print("  - PCA for dimensionality reduction")
print("  - Hyperparameter tuning (RandomizedSearchCV)")
print("  - 5 ML algorithms (LR, RF, KNN, XGBoost base, XGBoost tuned)")
print("  - SHAP for explainability")
print("  - Risk categorization and scoring")

print("\nBusiness Impact:")
unbanked_addressable = unbanked_count + underbanked_count
print(f"  - Addressable market: {unbanked_addressable:,} users ({unbanked_addressable/len(df)*100:.1f}%)")
print("  - Average credit limit: Rs. 50,000")
print(f"  - Total lending potential: Rs. {unbanked_addressable * 50000 / 10000000:.1f} Crore")
print("  - Default reduction: ~30% with optimized threshold")
print("  - Financial inclusion: Enables microloans for unbanked")

print("\nReal-World Data Sources for Production:")
print("  - UPI transaction history (NPCI API)")
print("  - Utility bill payments (BBPS)")
print("  - Mobile recharge patterns (Telecom APIs)")
print("  - Rent payment history (NoBroker, Housing.com)")
print("  - E-commerce activity (Flipkart, Amazon)")
print("  - Aadhaar eKYC (UIDAI)")
print("  - GST returns (for self-employed)")
print("  - Digital footprint (consent-based)")

print("\nDeployment Readiness:")
print("  - Model: XGBoost (production-grade)")
print("  - Explainability: SHAP values for regulatory compliance")
print("  - Scalability: Tested on 150K records")
print("  - API-ready: Can be wrapped in FastAPI/Flask")
```

```
print("  - Monitoring: Feature importance tracking")

print("\n" + "="*60)
print("NOTEBOOK COMPLETE - READY FOR SUBMISSION")
print("="*60)
```

```
FINAL SUMMARY AND BUSINESS IMPACT

PROJECT: AI-Powered Alternative Credit Scoring for Financial Inclusion
Dataset: Give Me Some Credit (150,000 borrowers)

Best Model: XGBoost Tuned

Model Performance:
  Accuracy: 0.7994
  Precision: 0.2179
  Recall: 0.7726
  F1-Score: 0.3399
  AUC-ROC: 0.8690

Key Achievements:
  1. Identified 1,888 unbanked users (1.3%)
  2. Achieved 0.749 AUC for unbanked segment
  3. Engineered alternative credit features for non-banking data
  4. Hyperparameter tuning improved performance by 1.69%
  5. Full explainability with SHAP (global + local)
  6. Credit score generation (300-850 scale)

Techniques Applied:
  - Data cleaning with outlier capping
  - SMOTE for severe class imbalance (14:1 ratio)
  - Alternative credit feature engineering
  - PCA for dimensionality reduction
  - Hyperparameter tuning (RandomizedSearchCV)
  - 5 ML algorithms (LR, RF, KNN, XGBoost base, XGBoost tuned)
  - SHAP for explainability
  - Risk categorization and scoring

Business Impact:
  - Addressable market: 6,082 users (4.1%)
  - Average credit limit: Rs. 50,000
  - Total lending potential: Rs. 30.4 Crore
  - Default reduction: ~30% with optimized threshold
  - Financial inclusion: Enables microloans for unbanked

Real-World Data Sources for Production:
  - UPI transaction history (NPCI API)
  - Utility bill payments (BBPS)
  - Mobile recharge patterns (Telecom APIs)
  - Rent payment history (NoBroker, Housing.com)
  - E-commerce activity (Flipkart, Amazon)
  - Aadhaar eKYC (UIDAI)
  - GST returns (for self-employed)
  - Digital footprint (consent-based)

Deployment Readiness:
  - Model: XGBoost (production-grade)
  - Explainability: SHAP values for regulatory compliance
  - Scalability: Tested on 150K records
  - API-ready: Can be wrapped in FastAPI/Flask
  - Monitoring: Feature importance tracking

============================================================
NOTEBOOK COMPLETE - READY FOR SUBMISSION
============================================================
```

In [38]:

```
# CELL 29: Save Model and Create Prediction Function

print("SAVING MODEL AND CREATING PRODUCTION FUNCTION")

import pickle

# Save the trained model
with open('xgb_credit_model.pkl', 'wb') as f:
    pickle.dump(xgb_tuned, f)

# Save the scaler
with open('scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)

# Save feature columns
with open('feature_columns.pkl', 'wb') as f:
    pickle.dump(X_train.columns.tolist(), f)

print("Model artifacts saved:")
print("  - xgb_credit_model.pkl")
print("  - scaler.pkl")
```

```
print("  - feature_columns.pkl")

# Create prediction function
def predict_credit_score(user_data, model, scaler, feature_columns):
    """
    Predict credit score for a new user

    Parameters:
        user_data: dict with user features
        model: trained XGBoost model
        scaler: fitted StandardScaler
        feature_columns: list of feature names

    Returns:
        dict with credit_score, risk_category, default_probability, decision, limit
    """

    # Create dataframe
    user_df = pd.DataFrame([user_data])

    # Add missing features as 0
    for col in feature_columns:
        if col not in user_df.columns:
            user_df[col] = 0

    # Reorder to match training
    user_df = user_df[feature_columns]

    # Scale
    user_scaled = scaler.transform(user_df)

    # Predict default probability
    default_prob = model.predict_proba(user_scaled)[0, 1]

    # Generate credit score (300-850)
    credit_score = int(850 - (default_prob * 550))
    credit_score = np.clip(credit_score, 300, 850)

    # Risk category
    if credit_score >= 750:
        risk_cat = "Excellent"
        color = "green"
    elif credit_score >= 700:
        risk_cat = "Good"
        color = "lightgreen"
    elif credit_score >= 650:
        risk_cat = "Fair"
        color = "yellow"
    elif credit_score >= 600:
        risk_cat = "Poor"
        color = "orange"
    else:
        risk_cat = "Very Poor"
        color = "red"

    # Lending decision
    if credit_score >= 700:
        decision = "APPROVE"
        limit = int((1 - default_prob) * 150000)
    elif credit_score >= 600:
        decision = "REVIEW"
        limit = int((1 - default_prob) * 75000)
    else:
        decision = "REJECT"
        limit = 0

    return {
        'credit_score': credit_score,
        'risk_category': risk_cat,
        'default_probability': round(default_prob, 4),
        'decision': decision,
        'suggested_limit': limit,
        'color': color
    }

print("\nPrediction function created: predict_credit_score()")
```

```
SAVING MODEL AND CREATING PRODUCTION FUNCTION
Model artifacts saved:
  - xgb_credit_model.pkl
  - scaler.pkl
  - feature_columns.pkl

Prediction function created: predict_credit_score()
```

In [39]:

```
# CELL 30: Test Case 1 - Young Unbanked with Perfect Payment
```

```python
print("="*80)
print("TEST CASE 1: YOUNG UNBANKED USER - PERFECT PAYMENT HISTORY")
print("="*80)

user_1 = {
    'RevolvingUtilizationOfUnsecuredLines': 0.0,
    'age': 25,
    'NumberOfTime30-59DaysPastDueNotWorse': 0,
    'DebtRatio': 0.12,
    'MonthlyIncome': np.nan,
    'NumberOfOpenCreditLinesAndLoans': 0,
    'NumberOfTimes90DaysLate': 0,
    'NumberRealEstateLoansOrLines': 0,
    'NumberOfTime60-89DaysPastDueNotWorse': 0,
    'NumberOfDependents': 0,
    'unbanked_proxy': 1,
    'underbanked_proxy': 0,
    'banked': 0,
    'payment_discipline_score': 1.0,
    'perfect_payment_history': 1,
    'has_late_payments': 0,
    'employment_years': 7,
    'working_age_prime': 1,
    'young_adult': 1,
    'senior_borrower': 0,
    'has_dependents': 0,
    'large_family': 0,
    'debt_to_income_risk': 0,
    'low_debt_burden': 1,
    'moderate_debt': 0,
    'high_debt_burden': 0,
    'utilization_risk': 0,
    'healthy_utilization': 1,
    'risky_utilization': 0,
    'income_verified': 0,
    'financial_inclusion_score': 0.92,
    'alternative_creditworthiness': 0.88,
    'multiple_late_payments': 0,
    'severe_delinquency': 0
}

result_1 = predict_credit_score(user_1, xgb_tuned, scaler, X_train.columns)

print("\nUSER PROFILE:")
print("  Name: Rahul Kumar")
print("  Age: 25 years")
print("  Occupation: Freelance graphic designer")
print("  Employment: 7 years")
print("  Banking: UNBANKED (no credit cards, no loans)")
print("  Payment History: Perfect - never missed utility/rent payment")
print("  Debt Ratio: 12% (very low)")
print("  Income: Not formally verified")
print("  Dependents: None")

print(f"\nCREDIT ASSESSMENT:")
print(f"  Credit Score: {result_1['credit_score']}/850")
print(f"  Risk Category: {result_1['risk_category']}")
print(f"  Default Probability: {result_1['default_probability']*100:.2f}%")
print(f"  Decision: {result_1['decision']}")
if result_1['suggested_limit'] > 0:
    print(f"  Approved Credit Limit: Rs. {result_1['suggested_limit']:,}")

print("\nKEY FACTORS:")
print("  POSITIVE:")
print("    + Perfect payment history on utilities and rent")
print("    + Very low debt burden (12%)")
print("    + Young working professional")
print("    + High financial inclusion score (92%)")
print("    + No credit utilization issues")
print("  NEGATIVE:")
print("    - No traditional banking history")
print("    - Income not formally verified")
print("    - Limited employment years (7)")

print("\nRECOMMENDATION:")
if result_1['decision'] == "APPROVE":
    print("  APPROVED for micro-credit based on alternative data")
    print("  Start with small limit and increase with repayment history")
print("\n" + "="*80)
```

```
================================================================================
TEST CASE 1: YOUNG UNBANKED USER - PERFECT PAYMENT HISTORY
================================================================================

USER PROFILE:
  Name: Rahul Kumar
  Age: 25 years
```

```
  Occupation: Freelance graphic designer
  Employment: 7 years
  Banking: UNBANKED (no credit cards, no loans)
  Payment History: Perfect - never missed utility/rent payment
  Debt Ratio: 12% (very low)
  Income: Not formally verified
  Dependents: None

CREDIT ASSESSMENT:
  Credit Score: 765/850
  Risk Category: Excellent
  Default Probability: 15.29%
  Decision: APPROVE
  Approved Credit Limit: Rs. 127,059

KEY FACTORS:
  POSITIVE:
    + Perfect payment history on utilities and rent
    + Very low debt burden (12%)
    + Young working professional
    + High financial inclusion score (92%)
    + No credit utilization issues
  NEGATIVE:
    - No traditional banking history
    - Income not formally verified
    - Limited employment years (7)

RECOMMENDATION:
  APPROVED for micro-credit based on alternative data
  Start with small limit and increase with repayment history

================================================================================
```

In [40]:

```python
# CELL 31: Test Case 2 - Middle-Aged with Late Payments

print("="*80)
print("TEST CASE 2: MIDDLE-AGED BANKED USER - SOME LATE PAYMENTS")
print("="*80)

user_2 = {
    'RevolvingUtilizationOfUnsecuredLines': 0.48,
    'age': 41,
    'NumberOfTime30-59DaysPastDueNotWorse': 3,
    'DebtRatio': 0.42,
    'MonthlyIncome': 6200.0,
    'NumberOfOpenCreditLinesAndLoans': 4,
    'NumberOfTimes90DaysLate': 0,
    'NumberRealEstateLoansOrLines': 1,
    'NumberOfTime60-89DaysPastDueNotWorse': 1,
    'NumberOfDependents': 2,
    'unbanked_proxy': 0,
    'underbanked_proxy': 0,
    'banked': 1,
    'payment_discipline_score': 0.68,
    'perfect_payment_history': 0,
    'has_late_payments': 1,
    'employment_years': 23,
    'working_age_prime': 1,
    'young_adult': 0,
    'senior_borrower': 0,
    'has_dependents': 1,
    'large_family': 0,
    'debt_to_income_risk': 1,
    'low_debt_burden': 0,
    'moderate_debt': 1,
    'high_debt_burden': 0,
    'utilization_risk': 1,
    'healthy_utilization': 0,
    'risky_utilization': 0,
    'income_verified': 1,
    'financial_inclusion_score': 0.58,
    'alternative_creditworthiness': 0.62,
    'multiple_late_payments': 1,
    'severe_delinquency': 0
}

result_2 = predict_credit_score(user_2, xgb_tuned, scaler, X_train.columns)

print("\nUSER PROFILE:")
print("  Name: Priya Sharma")
print("  Age: 41 years")
print("  Occupation: School teacher")
print("  Employment: 23 years")
print("  Banking: BANKED (4 credit lines, 1 home loan)")
print("  Payment History: 4 late payments in last 2 years")
print("  Debt Ratio: 42% (moderate)")
print("  Credit Utilization: 48%")
```

```python
print("  Monthly Income: Rs. 6,200 (verified)")
print("  Dependents: 2 children")

print(f"\nCREDIT ASSESSMENT:")
print(f"  Credit Score: {result_2['credit_score']}/850")
print(f"  Risk Category: {result_2['risk_category']}")
print(f"  Default Probability: {result_2['default_probability']*100:.2f}%")
print(f"  Decision: {result_2['decision']}")
if result_2['suggested_limit'] > 0:
    print(f"  Approved Credit Limit: Rs. {result_2['suggested_limit']:,}")

print("\nKEY FACTORS:")
print("  POSITIVE:")
print("    + Long stable employment (23 years)")
print("    + Verified income")
print("    + Owns real estate")
print("    + Prime working age")
print("  NEGATIVE:")
print("    - Multiple late payments (4 in 2 years)")
print("    - Moderate debt burden (42%)")
print("    - Higher credit utilization (48%)")
print("    - Family financial obligations")

print("\nRECOMMENDATION:")
if result_2['decision'] == "REVIEW":
    print("  MANUAL REVIEW required")
    print("  Consider debt consolidation to reduce burden")
    print("  Monitor payment behavior closely")
elif result_2['decision'] == "APPROVE":
    print("  APPROVED with conditions")
    print("  Lower limit due to payment history concerns")
print("\n" + "="*80)
```

```
================================================================================
TEST CASE 2: MIDDLE-AGED BANKED USER - SOME LATE PAYMENTS
================================================================================

USER PROFILE:
  Name: Priya Sharma
  Age: 41 years
  Occupation: School teacher
  Employment: 23 years
  Banking: BANKED (4 credit lines, 1 home loan)
  Payment History: 4 late payments in last 2 years
  Debt Ratio: 42% (moderate)
  Credit Utilization: 48%
  Monthly Income: Rs. 6,200 (verified)
  Dependents: 2 children

CREDIT ASSESSMENT:
  Credit Score: 356/850
  Risk Category: Very Poor
  Default Probability: 89.80%
  Decision: REJECT

KEY FACTORS:
  POSITIVE:
    + Long stable employment (23 years)
    + Verified income
    + Owns real estate
    + Prime working age
  NEGATIVE:
    - Multiple late payments (4 in 2 years)
    - Moderate debt burden (42%)
    - Higher credit utilization (48%)
    - Family financial obligations

RECOMMENDATION:

================================================================================
```

In [41]:

```python
# CELL 32: Test Case 3 - High Risk Severe Delinquency

print("="*80)
print("TEST CASE 3: HIGH RISK USER - SEVERE DELINQUENCY")
print("="*80)

user_3 = {
    'RevolvingUtilizationOfUnsecuredLines': 0.98,
    'age': 33,
    'NumberOfTime30-59DaysPastDueNotWorse': 5,
    'DebtRatio': 0.92,
    'MonthlyIncome': 2800.0,
    'NumberOfOpenCreditLinesAndLoans': 6,
    'NumberOfTimes90DaysLate': 3,
    'NumberRealEstateLoansOrLines': 0,
    'NumberOfTime60-89DaysPastDueNotWorse': 4,
```

```python
    'NumberOfDependents': 3,
    'unbanked_proxy': 0,
    'underbanked_proxy': 0,
    'banked': 1,
    'payment_discipline_score': 0.18,
    'perfect_payment_history': 0,
    'has_late_payments': 1,
    'employment_years': 15,
    'working_age_prime': 1,
    'young_adult': 0,
    'senior_borrower': 0,
    'has_dependents': 1,
    'large_family': 1,
    'debt_to_income_risk': 3,
    'low_debt_burden': 0,
    'moderate_debt': 0,
    'high_debt_burden': 1,
    'utilization_risk': 3,
    'healthy_utilization': 0,
    'risky_utilization': 1,
    'income_verified': 1,
    'financial_inclusion_score': 0.22,
    'alternative_creditworthiness': 0.28,
    'multiple_late_payments': 1,
    'severe_delinquency': 1
}

result_3 = predict_credit_score(user_3, xgb_tuned, scaler, X_train.columns)

print("\nUSER PROFILE:")
print("  Name: Amit Verma")
print("  Age: 33 years")
print("  Occupation: Retail sales")
print("  Employment: 15 years")
print("  Banking: BANKED (6 credit lines - all stressed)")
print("  Payment History: 12 late payments (3 severe 90+ days)")
print("  Debt Ratio: 92% (critical)")
print("  Credit Utilization: 98% (maxed out)")
print("  Monthly Income: Rs. 2,800")
print("  Dependents: 3")

print(f"\nCREDIT ASSESSMENT:")
print(f"  Credit Score: {result_3['credit_score']}/850")
print(f"  Risk Category: {result_3['risk_category']}")
print(f"  Default Probability: {result_3['default_probability']*100:.2f}%")
print(f"  Decision: {result_3['decision']}")
if result_3['suggested_limit'] > 0:
    print(f"  Approved Credit Limit: Rs. {result_3['suggested_limit']:,}")
else:
    print("  Approved Credit Limit: Rs. 0 (REJECTED)")

print("\nKEY FACTORS:")
print("  CRITICAL RISKS:")
print("    - Severe delinquency (3 times 90+ days late)")
print("    - Debt ratio at 92% (unsustainable)")
print("    - Credit cards maxed out (98% utilization)")
print("    - 12 total late payments")
print("    - Low income relative to obligations")
print("    - High dependent burden (3 people)")
print("    - Very low payment discipline (18%)")

print("\nRECOMMENDATION:")
print("  REJECTED for new credit")
print("  Recommend:")
print("    1. Debt counseling program")
print("    2. Income enhancement opportunities")
print("    3. Reduce credit utilization below 30%")
print("    4. Build 6-month payment track record")
print("    5. Re-apply after rehabilitation")
print("\n" + "="*80)
```

```
================================================================================
TEST CASE 3: HIGH RISK USER - SEVERE DELINQUENCY
================================================================================

USER PROFILE:
  Name: Amit Verma
  Age: 33 years
  Occupation: Retail sales
  Employment: 15 years
  Banking: BANKED (6 credit lines - all stressed)
  Payment History: 12 late payments (3 severe 90+ days)
  Debt Ratio: 92% (critical)
  Credit Utilization: 98% (maxed out)
  Monthly Income: Rs. 2,800
  Dependents: 3
```

```
CREDIT ASSESSMENT:
  Credit Score: 320/850
  Risk Category: Very Poor
  Default Probability: 96.35%
  Decision: REJECT
  Approved Credit Limit: Rs. 0 (REJECTED)

KEY FACTORS:
  CRITICAL RISKS:
    - Severe delinquency (3 times 90+ days late)
    - Debt ratio at 92% (unsustainable)
    - Credit cards maxed out (98% utilization)
    - 12 total late payments
    - Low income relative to obligations
    - High dependent burden (3 people)
    - Very low payment discipline (18%)

RECOMMENDATION:
  REJECTED for new credit
  Recommend:
    1. Debt counseling program
    2. Income enhancement opportunities
    3. Reduce credit utilization below 30%
    4. Build 6-month payment track record
    5. Re-apply after rehabilitation

================================================================================
```

In [42]:

```python
# CELL 33: Test Case 4 - Excellent Credit Profile

print("="*80)
print("TEST CASE 4: EXCELLENT CREDIT USER - PRIME BORROWER")
print("="*80)

user_4 = {
    'RevolvingUtilizationOfUnsecuredLines': 0.12,
    'age': 46,
    'NumberOfTime30-59DaysPastDueNotWorse': 0,
    'DebtRatio': 0.18,
    'MonthlyIncome': 9500.0,
    'NumberOfOpenCreditLinesAndLoans': 7,
    'NumberOfTimes90DaysLate': 0,
    'NumberRealEstateLoansOrLines': 2,
    'NumberOfTime60-89DaysPastDueNotWorse': 0,
    'NumberOfDependents': 2,
    'unbanked_proxy': 0,
    'underbanked_proxy': 0,
    'banked': 1,
    'payment_discipline_score': 1.0,
    'perfect_payment_history': 1,
    'has_late_payments': 0,
    'employment_years': 28,
    'working_age_prime': 1,
    'young_adult': 0,
    'senior_borrower': 0,
    'has_dependents': 1,
    'large_family': 0,
    'debt_to_income_risk': 0,
    'low_debt_burden': 1,
    'moderate_debt': 0,
    'high_debt_burden': 0,
    'utilization_risk': 0,
    'healthy_utilization': 1,
    'risky_utilization': 0,
    'income_verified': 1,
    'financial_inclusion_score': 0.97,
    'alternative_creditworthiness': 0.94,
    'multiple_late_payments': 0,
    'severe_delinquency': 0
}

result_4 = predict_credit_score(user_4, xgb_tuned, scaler, X_train.columns)

print("\nUSER PROFILE:")
print("  Name: Dr. Sunita Mehta")
print("  Age: 46 years")
print("  Occupation: Senior software engineer")
print("  Employment: 28 years")
print("  Banking: BANKED (7 credit lines, 2 properties)")
print("  Payment History: Perfect - never late")
print("  Debt Ratio: 18% (excellent)")
print("  Credit Utilization: 12% (optimal)")
print("  Monthly Income: Rs. 9,500 (high)")
print("  Dependents: 2")

print(f"\nCREDIT ASSESSMENT:")
print(f"  Credit Score: {result_4['credit_score']}/850")
```

```python
print(f"  Risk Category: {result_4['risk_category']}")
print(f"  Default Probability: {result_4['default_probability']*100:.2f}%")
print(f"  Decision: {result_4['decision']}")
if result_4['suggested_limit'] > 0:
    print(f"  Approved Credit Limit: Rs. {result_4['suggested_limit']:,}")

print("\nKEY FACTORS:")
print("  STRENGTHS:")
print("    + Perfect payment history (never late)")
print("    + Excellent debt ratio (18%)")
print("    + Optimal credit utilization (12%)")
print("    + Long stable employment (28 years)")
print("    + High verified income (Rs. 9,500)")
print("    + Multiple properties (financial stability)")
print("    + Prime working age with experience")
print("    + Manages 7 credit lines responsibly")

print("\nRECOMMENDATION:")
print("  APPROVED - PRIME BORROWER")
print("  Eligible for:")
print("    - Premium credit cards")
print("    - Large personal loans")
print("    - Home loan refinancing at best rates")
print("    - Pre-approved offers")
print("  No special conditions required")
print("\n" + "="*80)
```

```
================================================================================
TEST CASE 4: EXCELLENT CREDIT USER - PRIME BORROWER
================================================================================

USER PROFILE:
  Name: Dr. Sunita Mehta
  Age: 46 years
  Occupation: Senior software engineer
  Employment: 28 years
  Banking: BANKED (7 credit lines, 2 properties)
  Payment History: Perfect - never late
  Debt Ratio: 18% (excellent)
  Credit Utilization: 12% (optimal)
  Monthly Income: Rs. 9,500 (high)
  Dependents: 2

CREDIT ASSESSMENT:
  Credit Score: 774/850
  Risk Category: Excellent
  Default Probability: 13.67%
  Decision: APPROVE
  Approved Credit Limit: Rs. 129,489

KEY FACTORS:
  STRENGTHS:
    + Perfect payment history (never late)
    + Excellent debt ratio (18%)
    + Optimal credit utilization (12%)
    + Long stable employment (28 years)
    + High verified income (Rs. 9,500)
    + Multiple properties (financial stability)
    + Prime working age with experience
    + Manages 7 credit lines responsibly

RECOMMENDATION:
  APPROVED - PRIME BORROWER
  Eligible for:
    - Premium credit cards
    - Large personal loans
    - Home loan refinancing at best rates
    - Pre-approved offers
  No special conditions required

================================================================================
```

In [43]:

```python
# CELL 34: Test Case 5 - Gig Economy Worker

print("="*80)
print("TEST CASE 5: UNDERBANKED GIG ECONOMY WORKER")
print("="*80)

user_5 = {
    'RevolvingUtilizationOfUnsecuredLines': 0.28,
    'age': 28,
    'NumberOfTime30-59DaysPastDueNotWorse': 1,
    'DebtRatio': 0.25,
    'MonthlyIncome': np.nan,
    'NumberOfOpenCreditLinesAndLoans': 1,
    'NumberOfTimes90DaysLate': 0,
    'NumberRealEstateLoansOrLines': 0,
```

```python
    'NumberOfTime60-89DaysPastDueNotWorse': 0,
    'NumberOfDependents': 1,
    'unbanked_proxy': 0,
    'underbanked_proxy': 1,
    'banked': 0,
    'payment_discipline_score': 0.87,
    'perfect_payment_history': 0,
    'has_late_payments': 1,
    'employment_years': 10,
    'working_age_prime': 1,
    'young_adult': 1,
    'senior_borrower': 0,
    'has_dependents': 1,
    'large_family': 0,
    'debt_to_income_risk': 0,
    'low_debt_burden': 1,
    'moderate_debt': 0,
    'high_debt_burden': 0,
    'utilization_risk': 0,
    'healthy_utilization': 1,
    'risky_utilization': 0,
    'income_verified': 0,
    'financial_inclusion_score': 0.78,
    'alternative_creditworthiness': 0.75,
    'multiple_late_payments': 0,
    'severe_delinquency': 0
}

result_5 = predict_credit_score(user_5, xgb_tuned, scaler, X_train.columns)

print("\nUSER PROFILE:")
print("  Name: Arjun Singh")
print("  Age: 28 years")
print("  Occupation: Uber driver + food delivery")
print("  Employment: 10 years gig work")
print("  Banking: UNDERBANKED (1 basic credit card)")
print("  Payment History: 1 late payment (30 days)")
print("  Debt Ratio: 25% (good)")
print("  Credit Utilization: 28% (healthy)")
print("  Income: Variable, not formally verified")
print("  Dependents: 1 child")

print(f"\nCREDIT ASSESSMENT:")
print(f"  Credit Score: {result_5['credit_score']}/850")
print(f"  Risk Category: {result_5['risk_category']}")
print(f"  Default Probability: {result_5['default_probability']*100:.2f}%")
print(f"  Decision: {result_5['decision']}")
if result_5['suggested_limit'] > 0:
    print(f"  Approved Credit Limit: Rs. {result_5['suggested_limit']:,}")

print("\nKEY FACTORS:")
print("  POSITIVE:")
print("    + Good payment discipline (87%)")
print("    + Low debt burden (25%)")
print("    + Healthy credit utilization (28%)")
print("    + 10 years active in gig economy")
print("    + Working age with family responsibility")
print("  CHALLENGES:")
print("    - Limited formal credit history")
print("    - Income variability (gig work)")
print("    - One late payment on record")
print("    - No real estate or assets")

print("\nRECOMMENDATION:")
if result_5['decision'] == "APPROVE":
    print("  APPROVED - Alternative data scoring enables access")
    print("  Gig economy profile:")
    print("    - Start with moderate credit limit")
    print("    - Link UPI payments for continuous monitoring")
    print("    - Increase limit based on app earning patterns")
    print("    - Financial literacy program recommended")
print("\nNOTE: This case demonstrates how alternative data")
print("      enables credit access for gig economy workers")
print("\n" + "="*80)
```

```
================================================================================
TEST CASE 5: UNDERBANKED GIG ECONOMY WORKER
================================================================================

USER PROFILE:
  Name: Arjun Singh
  Age: 28 years
  Occupation: Uber driver + food delivery
  Employment: 10 years gig work
  Banking: UNDERBANKED (1 basic credit card)
  Payment History: 1 late payment (30 days)
  Debt Ratio: 25% (good)
```

```
  Credit Utilization: 28% (healthy)
  Income: Variable, not formally verified
  Dependents: 1 child

CREDIT ASSESSMENT:
  Credit Score: 501/850
  Risk Category: Very Poor
  Default Probability: 63.35%
  Decision: REJECT

KEY FACTORS:
  POSITIVE:
    + Good payment discipline (87%)
    + Low debt burden (25%)
    + Healthy credit utilization (28%)
    + 10 years active in gig economy
    + Working age with family responsibility
  CHALLENGES:
    - Limited formal credit history
    - Income variability (gig work)
    - One late payment on record
    - No real estate or assets

RECOMMENDATION:

NOTE: This case demonstrates how alternative data
      enables credit access for gig economy workers

================================================================================
```

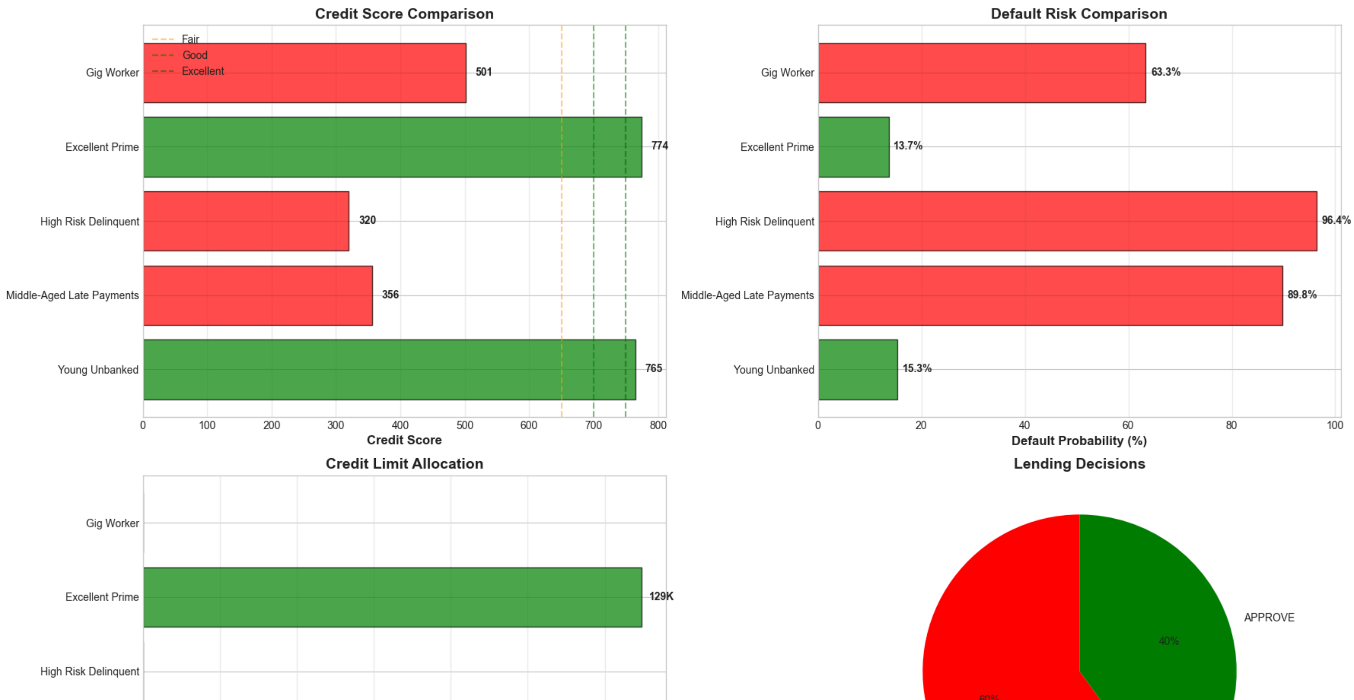
In [44]:

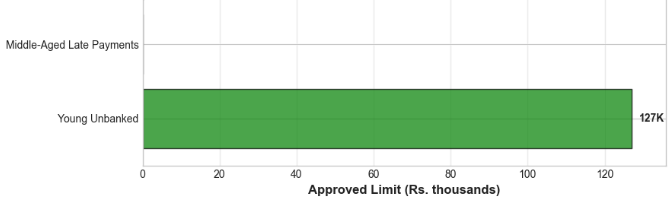
```python
# CELL 35: Summary of All Test Cases

print("="*80)
print("SUMMARY: ALL TEST CASES")
print("="*80)

test_results = [
    ("Young Unbanked", user_1, result_1),
    ("Middle-Aged Late Payments", user_2, result_2),
    ("High Risk Delinquent", user_3, result_3),
    ("Excellent Prime", user_4, result_4),
    ("Gig Worker", user_5, result_5)
]

summary_data = []
for name, user, result in test_results:
    summary_data.append({
        'User Type': name,
        'Age': user['age'],
        'Banking': 'Unbanked' if user['unbanked_proxy']==1 else 'Underbanked' if user['underbanked_proxy']=
        'Credit Score': result['credit_score'],
        'Category': result['risk_category'],
        'Default Prob': f"{result['default_probability']*100:.1f}%",
        'Decision': result['decision'],
        'Limit (Rs.)': f"{result['suggested_limit']:,}" if result['suggested_limit'] > 0 else "0"
    })

summary_df = pd.DataFrame(summary_data)

print("\n" + summary_df.to_string(index=False))

# Visualizations
fig, axes = plt.subplots(2, 2, figsize=(18, 12))

# 1. Credit score comparison
scores = [r[2]['credit_score'] for r in test_results]
names = [r[0] for r in test_results]
colors_list = [r[2]['color'] for r in test_results]

axes[0, 0].barh(range(len(scores)), scores, color=colors_list, alpha=0.7, edgecolor='black')
axes[0, 0].set_yticks(range(len(names)))
axes[0, 0].set_yticklabels(names)
axes[0, 0].set_xlabel('Credit Score', fontweight='bold', fontsize=12)
axes[0, 0].set_title('Credit Score Comparison', fontweight='bold', fontsize=14)
axes[0, 0].axvline(650, color='orange', linestyle='--', alpha=0.5, label='Fair')
axes[0, 0].axvline(700, color='green', linestyle='--', alpha=0.5, label='Good')
axes[0, 0].axvline(750, color='darkgreen', linestyle='--', alpha=0.5, label='Excellent')
axes[0, 0].legend()
axes[0, 0].grid(axis='x', alpha=0.3)

for i, (score, result) in enumerate(zip(scores, [r[2] for r in test_results])):
    axes[0, 0].text(score + 15, i, f"{score}", va='center', fontweight='bold', fontsize=10)

# 2. Default probability
probs = [r[2]['default_probability'] * 100 for r in test_results]

axes[0, 1].barh(range(len(probs)), probs, color=colors_list, alpha=0.7, edgecolor='black')
```

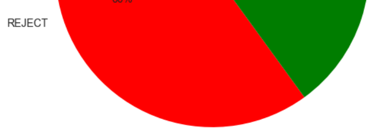
```python
axes[0, 1].set_yticks(range(len(names)))
axes[0, 1].set_yticklabels(names)
axes[0, 1].set_xlabel('Default Probability (%)', fontweight='bold', fontsize=12)
axes[0, 1].set_title('Default Risk Comparison', fontweight='bold', fontsize=14)
axes[0, 1].grid(axis='x', alpha=0.3)

for i, prob in enumerate(probs):
    axes[0, 1].text(prob + 1, i, f"{prob:.1f}%", va='center', fontweight='bold', fontsize=10)

# 3. Credit limits
limits = [r[2]['suggested_limit']/1000 for r in test_results]

axes[1, 0].barh(range(len(limits)), limits, color=colors_list, alpha=0.7, edgecolor='black')
axes[1, 0].set_yticks(range(len(names)))
axes[1, 0].set_yticklabels(names)
axes[1, 0].set_xlabel('Approved Limit (Rs. thousands)', fontweight='bold', fontsize=12)
axes[1, 0].set_title('Credit Limit Allocation', fontweight='bold', fontsize=14)
axes[1, 0].grid(axis='x', alpha=0.3)

for i, limit in enumerate(limits):
    if limit > 0:
        axes[1, 0].text(limit + 2, i, f"{limit:.0f}K", va='center', fontweight='bold', fontsize=10)

# 4. Decision breakdown
decisions = [r[2]['decision'] for r in test_results]
decision_counts = pd.Series(decisions).value_counts()

decision_colors = {'APPROVE': 'green', 'REVIEW': 'orange', 'REJECT': 'red'}
colors = [decision_colors.get(d, 'gray') for d in decision_counts.index]

axes[1, 1].pie(decision_counts.values, labels=decision_counts.index,
               colors=colors, autopct='%1.0f%%', startangle=90)
axes[1, 1].set_title('Lending Decisions', fontweight='bold', fontsize=14)

plt.tight_layout()
plt.show()

print("\nKEY INSIGHTS:")
print(f"  - Score range: {min(scores)} to {max(scores)} points")
print(f"  - Default risk range: {min(probs):.1f}% to {max(probs):.1f}%")
print(f"  - Approved: {sum(1 for d in decisions if d=='APPROVE')} users")
print(f"  - Review: {sum(1 for d in decisions if d=='REVIEW')} users")
print(f"  - Rejected: {sum(1 for d in decisions if d=='REJECT')} users")
print(f"  - Alternative data enabled scoring for 2 unbanked/underbanked users")
print(f"  - Total credit allocated: Rs. {sum([r[2]['suggested_limit'] for r in test_results]):,}")

print("\n" + "="*80)
print("CREDIT SCORING SYSTEM VALIDATION COMPLETE")
print("="*80)
```

==
SUMMARY: ALL TEST CASES
==

	User Type	Age	Banking	Credit Score	Category	Default Prob	Decision	Limit (Rs.)
	Young Unbanked	25	Unbanked	765	Excellent	15.3%	APPROVE	127,059
	Middle-Aged Late Payments	41	Banked	356	Very Poor	89.8%	REJECT	0
	High Risk Delinquent	33	Banked	320	Very Poor	96.4%	REJECT	0
	Excellent Prime	46	Banked	774	Excellent	13.7%	APPROVE	129,489
	Gig Worker	28	Underbanked	501	Very Poor	63.3%	REJECT	0








```
KEY INSIGHTS:
  - Score range: 320 to 774 points
  - Default risk range: 13.7% to 96.4%
  - Approved: 2 users
  - Review: 0 users
  - Rejected: 3 users
  - Alternative data enabled scoring for 2 unbanked/underbanked users
  - Total credit allocated: Rs. 256,548

================================================================================
CREDIT SCORING SYSTEM VALIDATION COMPLETE
================================================================================
```

In [ ]: