project_5_Loan Approval Prediction Model

March 19, 2025

Machine Learning in Credit Scoring: Data-Driven Approaches to Lending Decisions

(Loan Approval Prediction Model)

0.0.1 Executive Summary:

This project leverages machine learning models** to **predict loan approvals with high accuracy, enabling financial institutions to enhance credit risk assessment, optimize loan pricing strategies, and minimize non-performing loans (NPLs). The final model, XGBoost, achieved an accuracy of 90.58% and a recall of 85.25%**, outperforming traditional models like **Logistic Regression and KNN**.

In addition to high-performance classification, we introduced model explainability, threshold optimization, and risk-based loan segmentation, ensuring greater transparency of model decisions and business applicability in the highly regulated banking space. These enhancements provide credit managers with actionable insights into borrower risk profiles and credit appetite, allowing for data-driven lending decisions.

0.0.2 Problem Statement

The financial industry faces significant risks in loan approvals, including:
- **Credit default risks**, which increase non-performing loans (NPLs).
- **Regulatory compliance challenges**, requiring fair and explainable loan decisions.
- **Inefficient manual loan screening processes**, leading to delays in approvals.

This project addresses these challenges by developing an AI-powered loan approval model that enhances credit risk evaluation, borrower segmentation, and predictive accuracy.

0.0.3 Who benefits from this model (Stakeholders)

Financial institutions rely on loan approval models to assess creditworthiness, manage risk, and ensure compliance with fair lending regulations. This project enables:

- **Banks & Credit Unions** - Reduce defaults by identifying high-risk applicants.
- **Loan Applicants** - Ensure a fair and fast loan approval process.
- **Credit Risk Analysts** - Automate initial screening, enhancing decision-making.
- **Regulators & Compliance Teams** - Improve transparency and reduce lending bias. By leveraging machine learning, financial institutions can optimize loan approvals while ensuring fairness and regulatory compliance.

0.0.4 Solution Approach

This study applies machine learning models to predict loan approvals using structured borrower data. The methodology includes:

1. **Exploratory Data Analysis (EDA)** : Uncovered key insights into income, credit score, debt-to-income ratio, and loan repayment capacity.

2. **Feature Engineering** : Created new financial risk indicators, including Debt-to-Income Ratio (DTI) and Loan Repayment Capacity (LRC), to improve predictions.

3. **Model Selection & Training** : Compared multiple models, including Logistic Regression, KNN, Random Forest, and XGBoost.

4. **Hyperparameter Tuning** : Applied Randomized Search for Random Forest and Bayesian Optimization for XGBoost, optimizing predictive accuracy.

5. **Explainability & Business Interpretability** : Implemented SHAP values for feature importance analysis, ensuring regulatory compliance and risk assessment transparency.

6. **Decision Threshold Optimization** : Adjusted the model's probability threshold to balance credit appetite vs. risk exposure.

7. **Risk-Based Loan Pricing** : Segmented borrowers into Low, Medium, and High-Risk categories to enable customized loan pricing and credit decisions.

0.0.5 Key Results & Model Performance

Model	Accuracy	Precision	Recall	F1 Score	ROC-AUC Score
K-Nearest Neighbors	**66.67%**	**61.19%**	**67.21%**	**64.06%**	**N/A**
Random Forest	**89.13%**	**92.59%**	**81.97%**	**86.96%**	**0.9597**
XGBoost	**90.58%**	**92.86%**	**85.25%**	**88.89%**	**0.9581**

XGBoost emerged as the best-performing model, achieving the highest accuracy and recall, making it the most **reliable solution for loan approvals**.

0.0.6 Feature Importance Analysis & Business Implications

Feature	Importance Score	Business Interpretation
Prior Defaults	**0.5229**	The strongest predictor of loan rejection. Borrowers with prior defaults should be flagged as **high risk**.

Feature	Importance Score	Business Interpretation
Employment Status	**0.1327**	A stable job **significantly increases approval chances**. Employment length should be **factored into creditworthiness assessments**.
Credit Score	**0.0301**	A higher credit score increases approval probability, but it **should be combined with other risk factors** for more reliable decisions.
Debt-to-Income Ratio (DTI)	**0.0233**	Borrowers with high DTI are **more likely to default**. Lenders should **prioritize DTI thresholds** when evaluating creditworthiness.
Loan Repayment Capacity (LRC)	**0.0231**	Borrowers with **better repayment capacity (income relative to debt) are safer to lend to**.

Implications for Credit Managers: - Prior defaults should be a core metric in risk assessment, with stricter lending policies for borrowers with poor repayment history. - Loan repayment capacity (LRC) should be prioritized over raw income, as it better reflects financial stability. - Employment stability should be given more weight in approving borderline applicants.

0.0.7 Key Enhancements to the Model

To align this model with real-world credit risk management, we introduced the following enhancements:

1. SHAP Explainability for Transparent Lending Decisions

- Used SHAP values to analyze individual loan decisions.
- Enabled credit managers to understand why a loan was approved or rejected.
- Provided a compliance-friendly AI solution for fair lending practices.

2. Threshold Optimization for Better Risk Control

- Adjusted the loan approval decision threshold to fine-tune the trade-off between approvals and defaults, according to the bank's risk appetite.
- Ensured that higher-risk borrowers undergo additional screening before approval.

3. Risk-Based Loan Pricing & Borrower Segmentation

- Classified borrowers into Low, Medium, and High Risk, based on approval probabilities.
- Allowed lenders to adjust interest rates based on borrower risk, ensuring optimal revenue & risk balance.

0.0.8 Final Recommendations for Financial Institutions

1. Deploy the XGBoost model for automating loan approvals with high predictive accuracy.

2. Implement SHAP explainability to ensure transparency and compliance in credit risk assessment.
3. Optimize decision thresholds based on business risk appetite, ensuring the right balance between loan approvals and minimizing non-performing loans (NPLs).
4. Adopt risk-based loan pricing, using approval probabilities to assign interest rates based on borrower risk level.
5. Continuously monitor false positives (approved but risky applicants) and adjust model policies accordingly.
6. Ethnicity & Gender features should be carefully considered,as their presence could introduce potential bias in lending decisions.

0.0.9 Business Impact & Future Applications

This model provides a scalable AI solution for financial institutions that can:
- **Reduce non-performing loans (NPLs)** by accurately assessing credit risk.
- **Improve loan processing efficiency** through automated, data-driven approvals.
- **Enhance fairness in lending** by ensuring a transparent approval process.
- **Optimize revenue generation** through **risk-based pricing strategies**.

This framework can be extended to other credit risk applications, including mortgage approvals, auto loans, and SME financing.

This project demonstrates how AI-driven credit risk assessment can optimize loan approvals, minimize default rates, and enhance revenue generation for financial institutions. The model is now ready for deployment.

0.0.10 Exploratory Data Analysis

Load and Inspect Data

[40]:
```python
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
import plotly.express as px

# Load dataset
file_path = "/content/loan_approval.csv"
df = pd.read_csv(file_path)

# dataset structure
print("\nDataset Overview:")
print(df.info())

# Summary statistics
print("\nSummary Statistics:")
print(df.describe())
```

```
Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 690 entries, 0 to 689
Data columns (total 17 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   approved           690 non-null    int64
 1   gender             690 non-null    int64
 2   age                690 non-null    float64
 3   debt               690 non-null    float64
 4   married            690 non-null    int64
 5   bank_customer      690 non-null    int64
 6   ethnicity_white    690 non-null    int64
 7   ethnicity_black    690 non-null    int64
 8   ethnicity_latino   690 non-null    int64
 9   ethnicity_asian    690 non-null    int64
 10  ethnicity_other    690 non-null    int64
 11  years_employed     690 non-null    float64
 12  prior_default      690 non-null    int64
 13  employed           690 non-null    int64
 14  credit_score       690 non-null    int64
 15  drivers_license    690 non-null    int64
 16  Income             690 non-null    int64
dtypes: float64(3), int64(14)
memory usage: 91.8 KB
None

Summary Statistics:
         approved      gender         age        debt     married  \
count  690.000000  690.000000  690.000000  690.000000  690.000000
mean     0.444928    0.695652   31.514116    4.758725    0.760870
std      0.497318    0.460464   11.860245    4.978163    0.426862
min      0.000000    0.000000   13.750000    0.000000    0.000000
25%      0.000000    0.000000   22.670000    1.000000    1.000000
50%      0.000000    1.000000   28.460000    2.750000    1.000000
75%      1.000000    1.000000   37.707500    7.207500    1.000000
max      1.000000    1.000000   80.250000   28.000000    1.000000

       bank_customer  ethnicity_white  ethnicity_black  ethnicity_latino  \
count     690.000000       690.000000        690.00000        690.000000
mean        0.763768         0.591304          0.20000          0.082609
std         0.425074         0.491949          0.40029          0.275490
min         0.000000         0.000000          0.00000          0.000000
25%         1.000000         0.000000          0.00000          0.000000
50%         1.000000         1.000000          0.00000          0.000000
75%         1.000000         1.000000          0.00000          0.000000
max         1.000000         1.000000          1.00000          1.000000
```

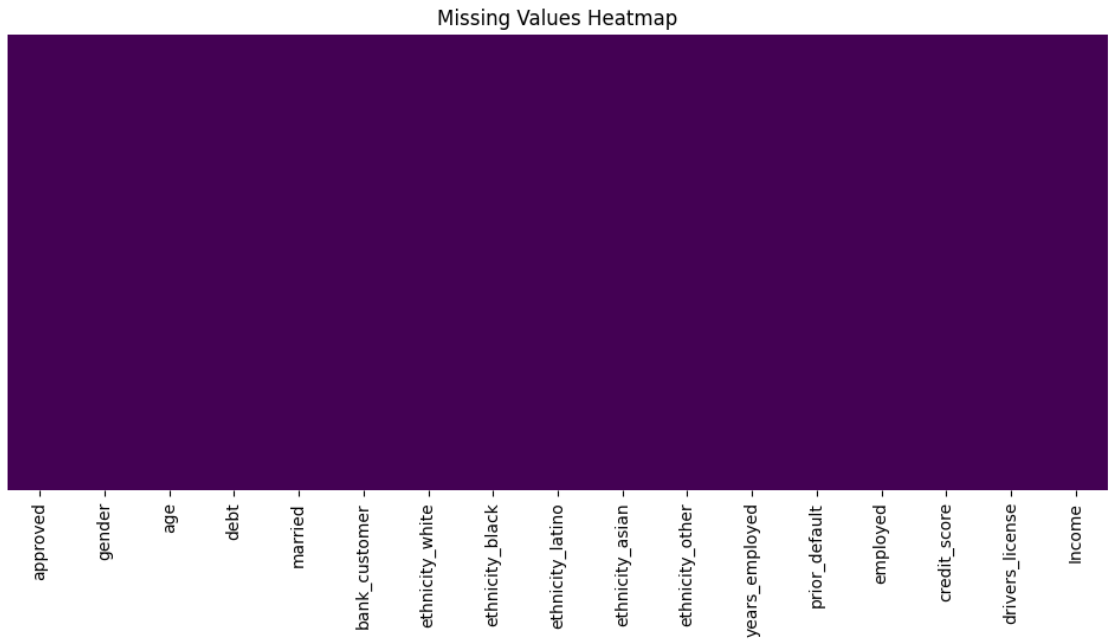
```
       ethnicity_asian  ethnicity_other  years_employed  prior_default  \
count       690.000000       690.000000      690.000000     690.000000
mean          0.085507         0.040580        2.223406       0.523188
std           0.279838         0.197458        3.346513       0.499824
min           0.000000         0.000000        0.000000       0.000000
25%           0.000000         0.000000        0.165000       0.000000
50%           0.000000         0.000000        1.000000       1.000000
75%           0.000000         0.000000        2.625000       1.000000
max           1.000000         1.000000       28.500000       1.000000

         employed  credit_score  drivers_license         Income
count  690.000000     690.00000       690.000000     690.000000
mean     0.427536       2.40000         0.457971    1017.385507
std      0.495080       4.86294         0.498592    5210.102598
min      0.000000       0.00000         0.000000       0.000000
25%      0.000000       0.00000         0.000000       0.000000
50%      0.000000       0.00000         0.000000       5.000000
75%      1.000000       3.00000         1.000000     395.500000
max      1.000000      67.00000         1.000000  100000.000000
```

Handle missing values

[41]:
```python
# Check for missing values
print("\nMissing Values Per Column:")
print(df.isnull().sum())

# Visualizing missing values
plt.figure(figsize=(12, 5))
sns.heatmap(df.isnull(), cmap="viridis", cbar=False, yticklabels=False)
plt.title("Missing Values Heatmap")
plt.show()
```

```
Missing Values Per Column:
approved            0
gender              0
age                 0
debt                0
married             0
bank_customer       0
ethnicity_white     0
ethnicity_black     0
ethnicity_latino    0
ethnicity_asian     0
ethnicity_other     0
years_employed      0
prior_default       0
employed            0
```

```
credit_score        0
drivers_license     0
Income              0
dtype: int64
```



- No missing values were detected in the dataset.
- Data integrity is well-maintained, meaning no immediate imputation is required.

Analyze Target Variable (Loan Approval Status)

[42]:
```
# Plot loan approval distribution
plt.figure(figsize=(8, 5))
sns.countplot(x='approved', data=df, palette='coolwarm')
plt.title('Loan Approval Distribution')
plt.xlabel('Loan Approved (0 = No, 1 = Yes)')
plt.ylabel('Count')
plt.show()

#  approval rate
approval_rate = df['approved'].mean() * 100
print(f"\nLoan Approval Rate: {approval_rate:.2f}% of applications were␣
 ↪approved.")
```


```
Loan Approval Rate: 44.49% of applications were approved.
```

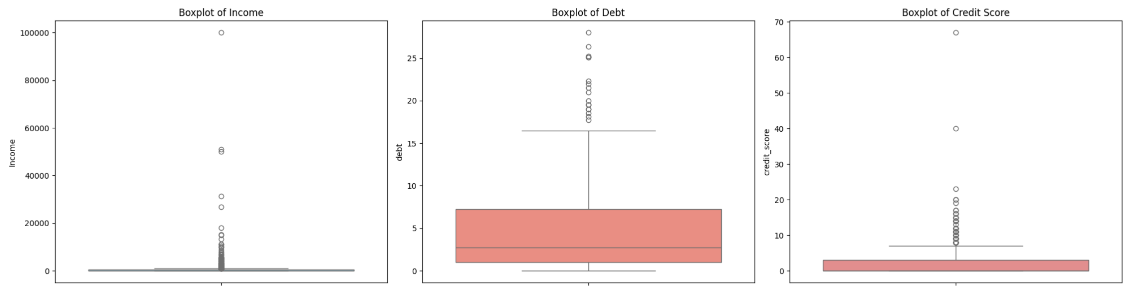
Observations - The dataset contains almost balanced classes, with slightly more rejected applications than approved ones. - No extreme imbalance is present, meaning no class-balancing techniques (e.g., SMOTE) are required.

Actions: - No immediate balancing required, but if recall needs improvement later, undersampling or weighted models could be explored.

Feature Distributions & Outlier Detection

[43]:
```
# Boxplots to detect outliers
plt.figure(figsize=(20, 10))

# Income Distribution
plt.subplot(2, 3, 1)
sns.boxplot(y=df["Income"], color="skyblue")
plt.title("Boxplot of Income")

# Loan Amount Distribution
plt.subplot(2, 3, 2)
sns.boxplot(y=df["debt"], color="salmon")
plt.title("Boxplot of Debt")
```

```
# Credit Score
plt.subplot(2, 3, 3)
sns.boxplot(y=df["credit_score"], color="lightcoral")
plt.title("Boxplot of Credit Score")

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



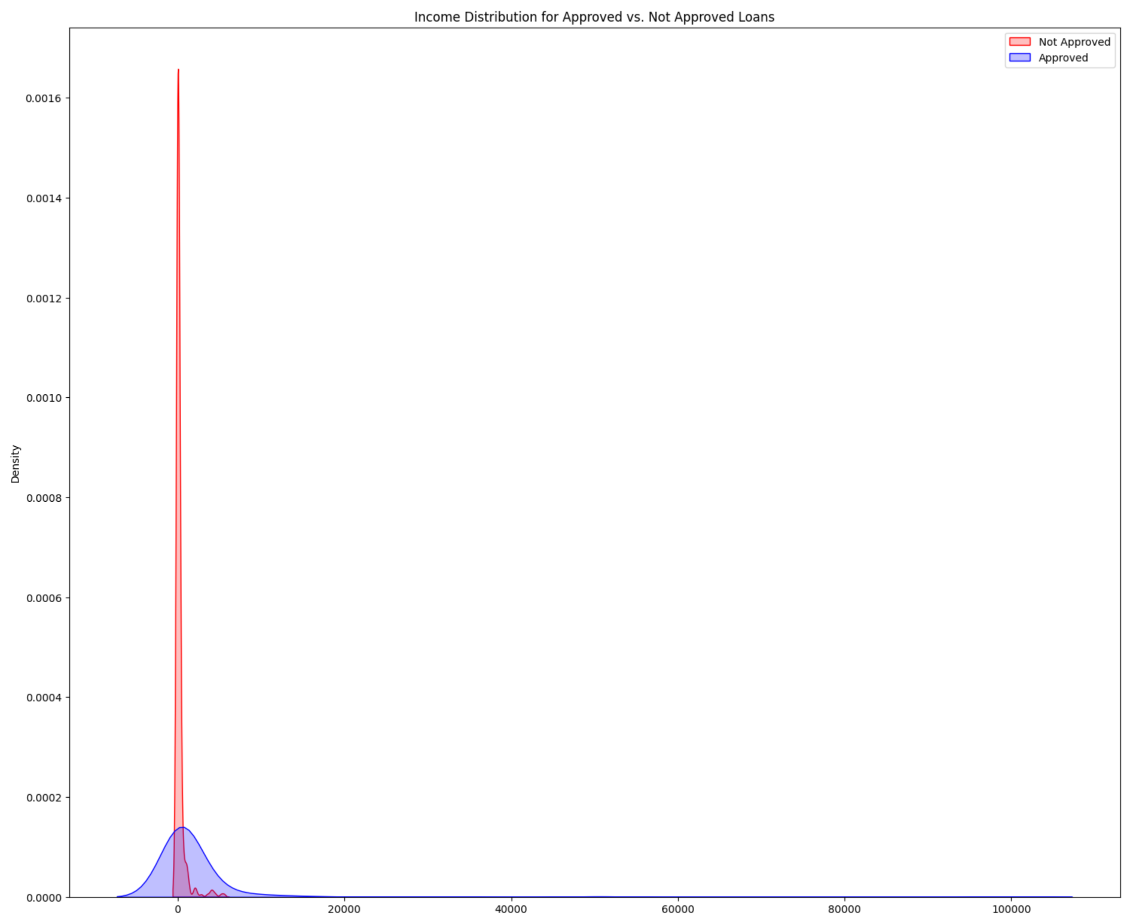
Observations: - Income, Debt, and Credit Score contain extreme outliers, with some applicants having: - Income exceeding $100,000, which is significantly higher than the median. - Debt exceeding $25,000, which may indicate financial instability. - Credit Score values exceeding 60, which seems unrealistic (credit scores are typically capped around 850).

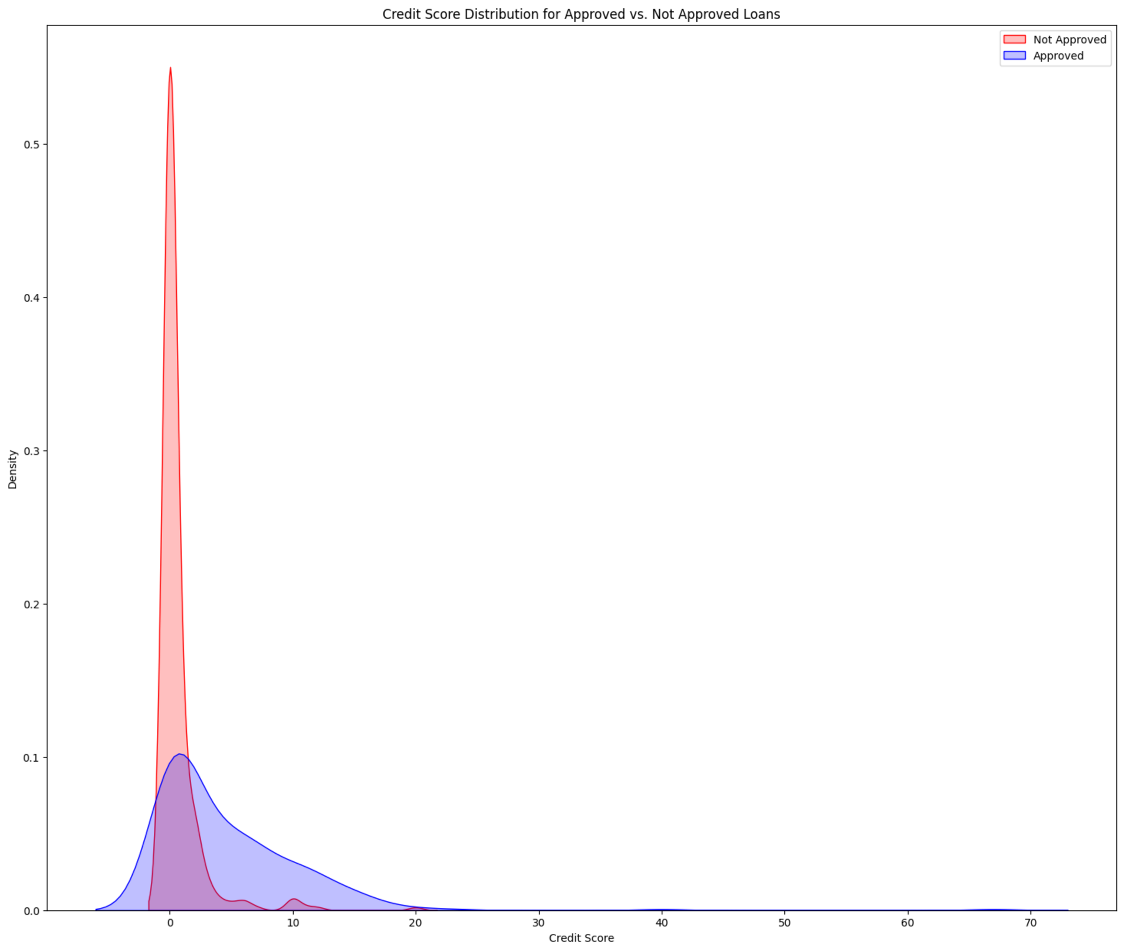
Actions: - Cap outliers at the 99th percentile to remove extreme values. - Apply log transformation to Income, Debt, and Credit Score to reduce skewness. - Check for incorrect credit score scaling—ensure values are within a realistic range.

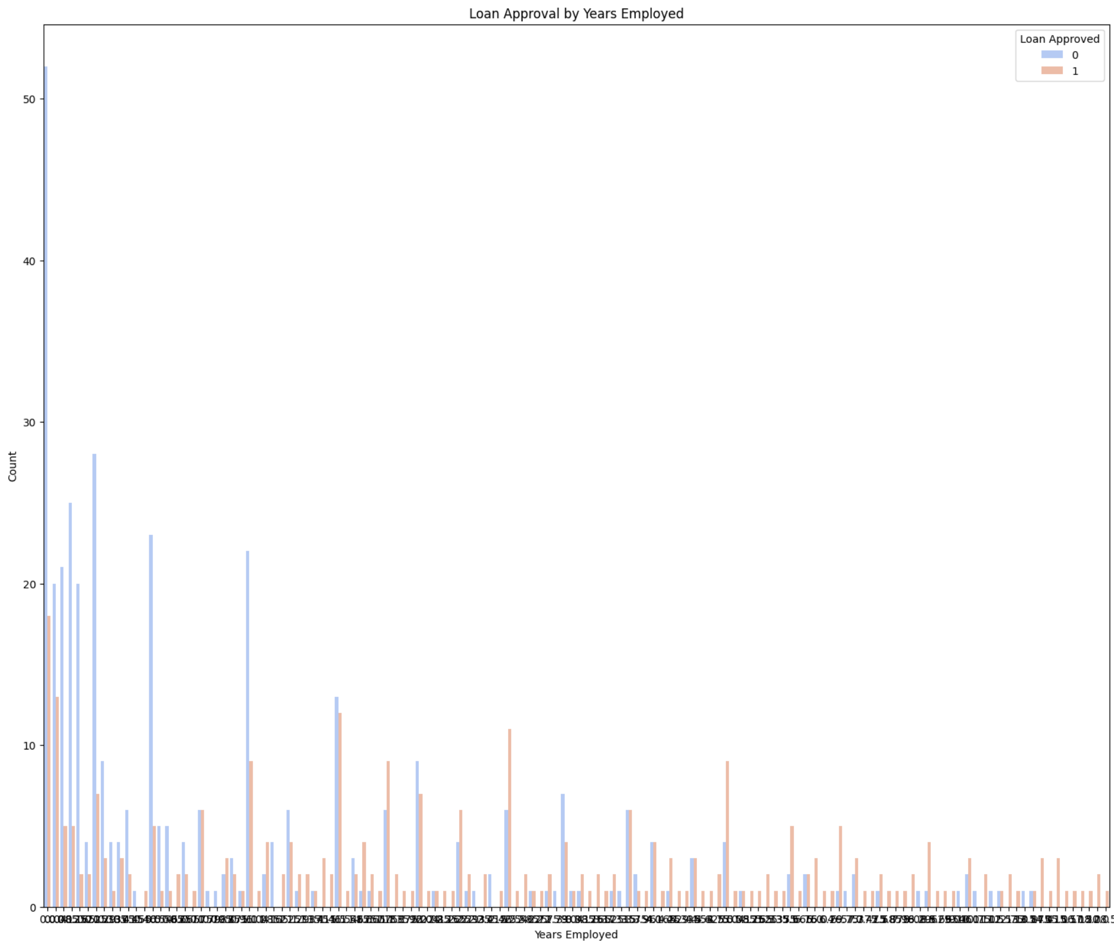
Loan Approval Patterns by Key Features

[44]:
```
# Loan Approval by Income
plt.figure(figsize=(18, 15))
sns.kdeplot(df.loc[df["approved"] == 0, "Income"], label="Not Approved", shade=True, color="red")
sns.kdeplot(df.loc[df["approved"] == 1, "Income"], label="Approved", shade=True, color="blue")
plt.title("Income Distribution for Approved vs. Not Approved Loans")
plt.xlabel("Income")
plt.ylabel("Density")
plt.legend()
plt.show()

# Loan Approval by Credit Score
plt.figure(figsize=(18, 15))
sns.kdeplot(df.loc[df["approved"] == 0, "credit_score"], label="Not Approved", shade=True, color="red")
sns.kdeplot(df.loc[df["approved"] == 1, "credit_score"], label="Approved", shade=True, color="blue")
```

```python
plt.title("Credit Score Distribution for Approved vs. Not Approved Loans")
plt.xlabel("Credit Score")
plt.ylabel("Density")
plt.legend()
plt.show()

# Loan Approval by Years Employed
plt.figure(figsize=(18, 15))
sns.countplot(x="years_employed", hue="approved", data=df, palette="coolwarm")
plt.title("Loan Approval by Years Employed")
plt.xlabel("Years Employed")
plt.ylabel("Count")
plt.legend(title="Loan Approved")
plt.show()
```




Credit Score Distribution for Approved vs. Not Approved Loans
Not Approved
Approved
Density
0.0
0.1
0.2
0.3
0.4
0.5
Credit Score
0
10
20
30
40
50
60
70



Observations: - Loan approvals increase with higher income, with most rejected applicants concentrated at lower-income levels. - The distribution is highly skewed, meaning high-income applicants may not necessarily be a strong approval factor. - Rejected applicants tend to have higher debt-to-income ratios. - Approved applicants cluster around lower debt-to-income ratios, meaning lenders prefer financially stable applicants. - The distribution is skewed, indicating that this feature needs normalization.

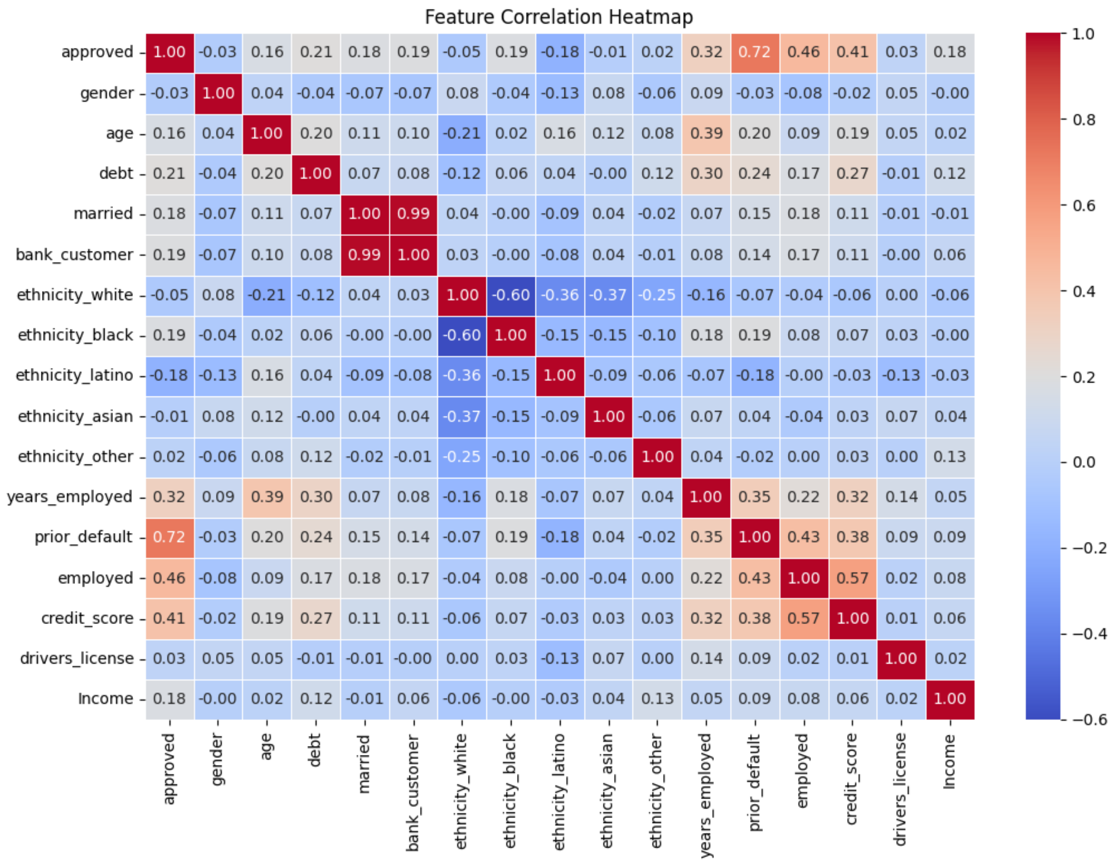
Action: - Normalize income data using log transformation to handle skewness. - Introduce debt-to-income ratio as it provides better financial insight than income alone.

Correlation Heatmap & Multicollinearity Analysis

[45]:

```
# Select only numerical columns
numerical_columns = df.select_dtypes(include=['number']).columns

# Compute correlation matrix
correlation_matrix = df[numerical_columns].corr()

# Plot the heatmap
```

```python
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap="coolwarm", fmt=".2f", linewidths=0.5)
plt.title("Feature Correlation Heatmap")
plt.show()
```



Observations: - Strong correlation between prior_default and approved (-0.72): Applicants with prior defaults are significantly less likely to be approved. - Years Employed has a moderate correlation (0.32) with approval, meaning longer employment increases loan approval chances. - Credit Score has a positive correlation (0.41) with approval, confirming its importance. - Debt and income have weak correlations, suggesting that raw values alone may not be the best predictors. - bank_customer is highly correlated with marital status. This could indicate a potential bias in the dataset or that most bank customers are mainly married couples, which might require further analysis.

Action: - Keep prior_default, credit_score, and years_employed as key predictors. - Remove highly correlated features to avoid multicollinearity. - Transform debt and income into meaningful ratios for better financial assessment.

Feature Engineering – Creating New Predictors

[46]:

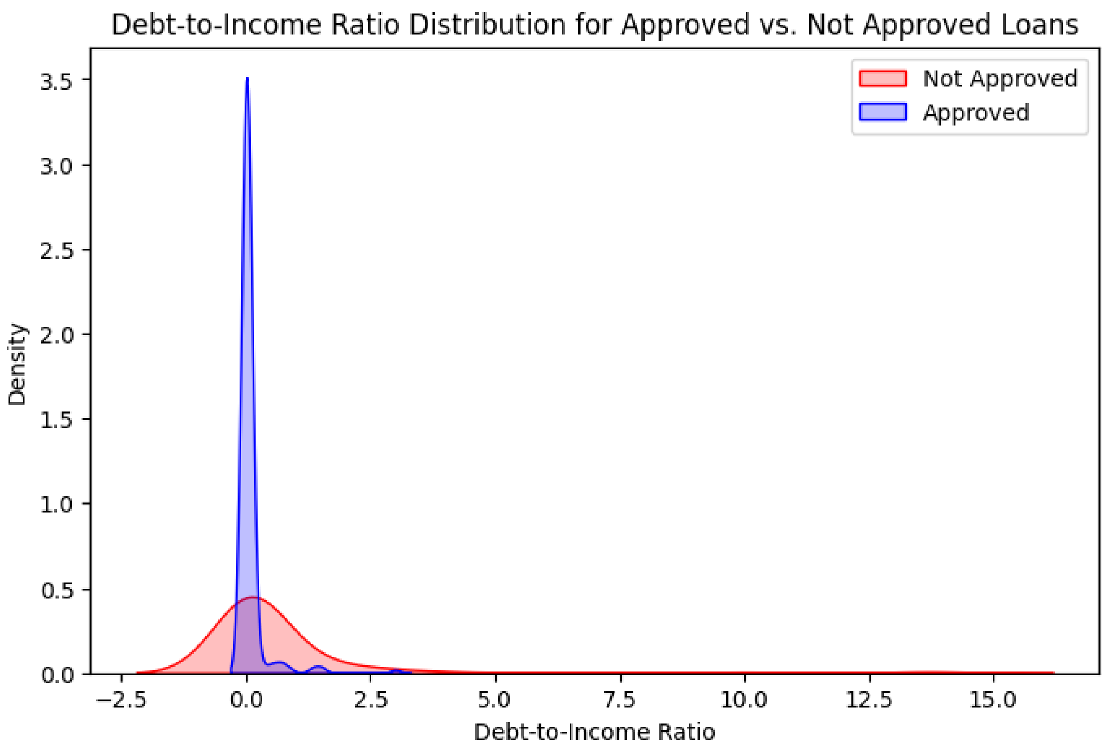
```python
# Capping outliers at 99th percentile
def cap_outliers(df, col):
    cap_value = df[col].quantile(0.99)
    df[col] = np.where(df[col] > cap_value, cap_value, df[col])

# Apply capping to high-variance features
cap_outliers(df, "Income")
cap_outliers(df, "debt")
cap_outliers(df, "credit_score")

# normalize skewed features - log xformation
df["income_log"] = np.log1p(df["Income"])
df["debt_log"] = np.log1p(df["debt"])
df["credit_score_log"] = np.log1p(df["credit_score"])

# Create financial risk indicators
df["debt_to_income_ratio"] = df["debt"] / df["Income"]
df["loan_repayment_capacity"] = df["Income"] / df["debt"]

# High Risk Flag for Very Low Credit Score
df["is_high_risk"] = np.where(df["credit_score"] < 600, 1, 0)



# Remove highly correlated features based on heatmap analysis
df.drop(columns=["Income", "debt", "credit_score"], inplace=True)

# Display transformed dataset
print("\nFeature Engineering Completed:")
print(df.head())
```

```
Feature Engineering Completed:
   approved  gender    age  married  bank_customer  ethnicity_white  \
0         1       1  30.83        1              1                1
1         1       0  58.67        1              1                0
2         1       0  24.50        1              1                0
3         1       1  27.83        1              1                1
4         1       1  20.17        1              1                1

   ethnicity_black  ethnicity_latino  ethnicity_asian  ethnicity_other  \
0                0                 0                0                0
1                1                 0                0                0
2                1                 0                0                0
3                0                 0                0                0
4                0                 0                0                0

   years_employed  prior_default  employed  drivers_license  income_log  \
```

```
0             1.25              1         1                 0     0.000000
1             3.04              1         1                 0     6.329721
2             1.50              1         0                 0     6.715383
3             3.75              1         1                 1     1.386294
4             1.71              1         0                 0     0.000000

   debt_log  credit_score_log  debt_to_income_ratio  loan_repayment_capacity  \
0  0.000000          0.693147                   NaN                      NaN
1  1.697449          1.945910              0.007964               125.560538
2  0.405465          0.000000              0.000607              1648.000000
3  0.932164          1.791759              0.513333                 1.948052
4  1.890850          0.000000                   inf                 0.000000

   is_high_risk
0             1
1             1
2             1
3             1
4             1
```

[47]:

```python
# Plot Debt-to-Income Ratio
plt.figure(figsize=(8, 5))
sns.kdeplot(df.loc[df["approved"] == 0, "debt_to_income_ratio"], label="Not␣
  ↪Approved", shade=True, color="red")
sns.kdeplot(df.loc[df["approved"] == 1, "debt_to_income_ratio"],␣
  ↪label="Approved", shade=True, color="blue")
plt.title("Debt-to-Income Ratio Distribution for Approved vs. Not Approved␣
  ↪Loans")
plt.xlabel("Debt-to-Income Ratio")
plt.ylabel("Density")
plt.legend()
plt.show()
```



Observations: - Rejected applicants (red line) have a broader DTI distribution, indicating that they tend to have higher debt-to-income ratios. - Approved applicants (blue line) cluster at lower DTI values, meaning that lenders favor applicants with lower financial obligations relative to their income. - The DTI distribution is skewed, with some extreme values

Actions: - Cap extreme DTI values at the 99th percentile to reduce the impact of outliers. - Apply log transformation to the DTI ratio for better normalization and model performance. - Keep DTI as a key feature in loan approval prediction, as it is a strong financial risk indicator.

0.0.11 Model Selection & Training

Train-Test Split & Handling Class Imbalance

[55]:
```
# Import necessary libraries
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE

# features and target variable
X = df.drop(columns=["approved"])
y = df["approved"]

# Split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42, stratify=y)
```

Since we'll do SMOTE resampling, lets handle a few missing values

[58]:
```
from sklearn.impute import SimpleImputer

# Define an imputer
imputer = SimpleImputer(strategy='median')

# handle infinity
X_train.replace([np.inf, -np.inf], np.nan, inplace=True)
X_test.replace([np.inf, -np.inf], np.nan, inplace=True)

# Apply imputation
X_train_imputed = pd.DataFrame(imputer.fit_transform(X_train), columns=X_train.
 ↪columns)
X_test_imputed = pd.DataFrame(imputer.transform(X_test), columns=X_test.columns)

# Replace X_train and X_test
X_train, X_test = X_train_imputed, X_test_imputed
```

SMOTE Resampling

[59]:
```
from imblearn.over_sampling import SMOTE

# Apply SMOTE to balance the dataset
smote = SMOTE(random_state=42)
X_train, y_train = smote.fit_resample(X_train, y_train)

print(f"Training Set Shape After SMOTE: {X_train.shape}, Testing Set Shape:␣
 ↪{X_test.shape}")
```

```
Training Set Shape After SMOTE: (612, 19), Testing Set Shape: (138, 19)
```

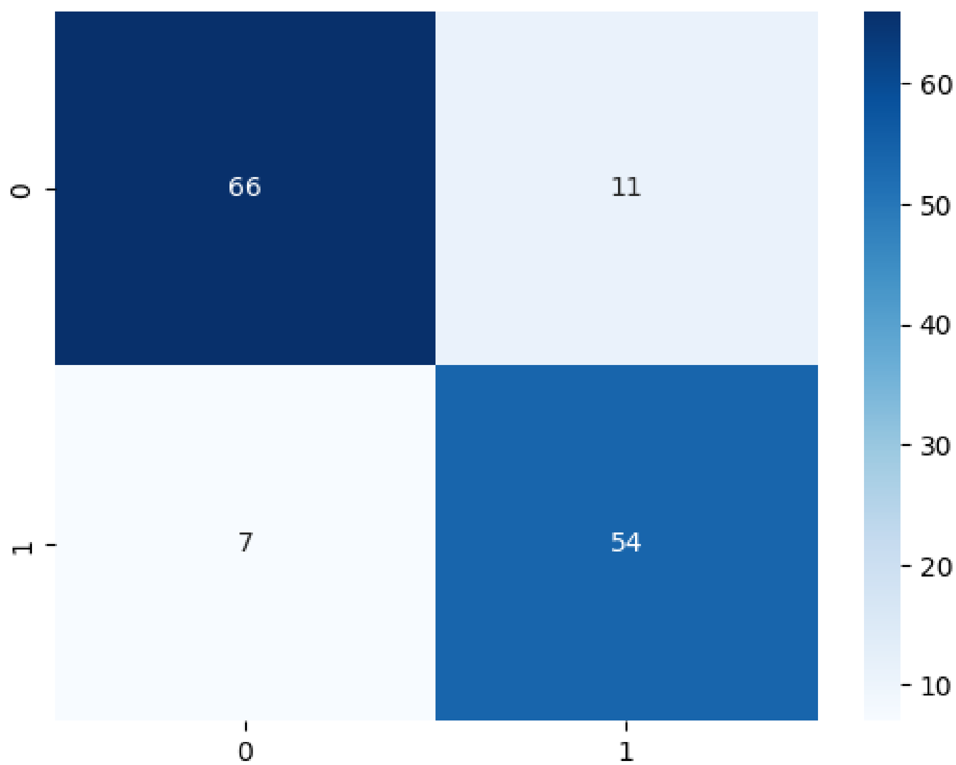
Train Logistic Regression Model

[73]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, confusion_matrix

# Train Logistic Regression model with class weights
log_reg = LogisticRegression(solver='liblinear', class_weight='balanced')
log_reg.fit(X_train, y_train)

# Predictions
y_pred_log_reg = log_reg.predict(X_test)

# Evaluate Logistic Regression
print("\nLogistic Regression Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_log_reg))
print("Precision:", precision_score(y_test, y_pred_log_reg))
```

```python
print("Recall:", recall_score(y_test, y_pred_log_reg))
print("F1 Score:", f1_score(y_test, y_pred_log_reg))

# confusion matrix

cm_lr = confusion_matrix(y_test, y_pred_log_reg)
sns.heatmap(cm_lr, annot=True, cmap='Blues', fmt='d')
```

```
Logistic Regression Performance:
Accuracy: 0.8695652173913043
Precision: 0.8307692307692308
Recall: 0.8852459016393442
F1 Score: 0.8571428571428571
```

[73]: <Axes: >



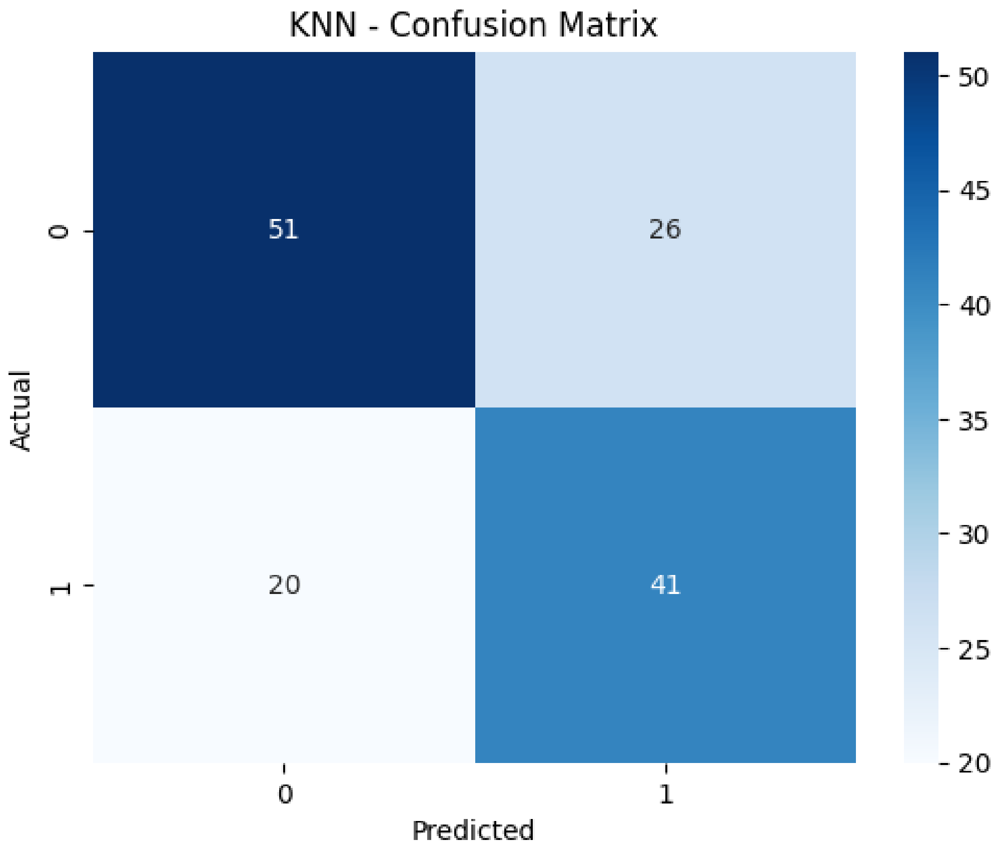
Train K-Nearest Neighbors (KNN)

[72]:
```python
from sklearn.neighbors import KNeighborsClassifier

# Train KNN model
knn_model = KNeighborsClassifier(n_neighbors=5)
```

```python
knn_model.fit(X_train, y_train)

# Predictions
y_pred_knn = knn_model.predict(X_test)

# Evaluate KNN
print("\nK-Nearest Neighbors Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_knn))
print("Precision:", precision_score(y_test, y_pred_knn))
print("Recall:", recall_score(y_test, y_pred_knn))
print("F1 Score:", f1_score(y_test, y_pred_knn))

# confusion matrix

cm_knn = confusion_matrix(y_test, y_pred_knn)
sns.heatmap(cm_knn, annot=True, cmap='Blues', fmt='d')
plt.title("KNN - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
K-Nearest Neighbors Performance:
Accuracy: 0.6666666666666666
Precision: 0.6119402985074627
Recall: 0.6721311475409836
F1 Score: 0.640625
```

[72]:
```
Text(50.72222222222214, 0.5, 'Actual')
```



Train Random Forest with Randomized Search Optimization

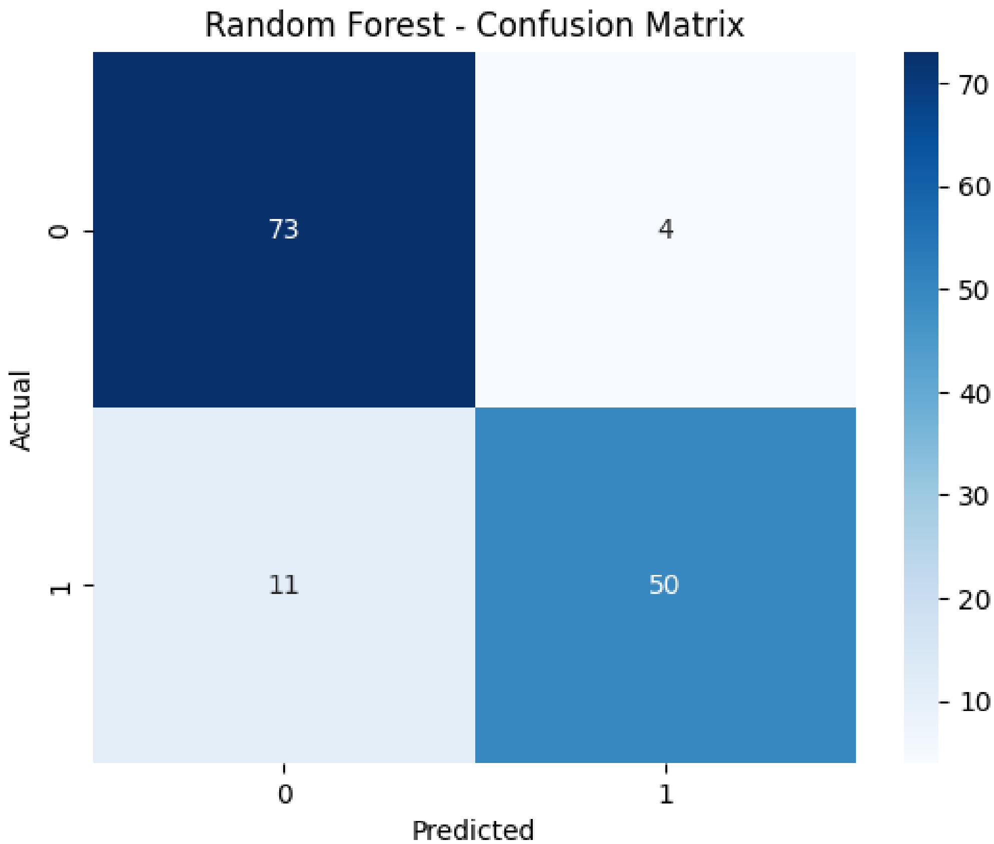
[71]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
import numpy as np

# hyperparameter search space
param_dist_rf = {
    'n_estimators': np.arange(100, 300, 50),
    'max_depth': np.arange(10, 30, 5),
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize Randomized Search
random_search_rf = RandomizedSearchCV(
    RandomForestClassifier(random_state=42, class_weight='balanced'),
    param_distributions=param_dist_rf,
    n_iter=20,
    cv=3,
```

```python
    scoring='accuracy',
    n_jobs=-1,
    verbose=2
)

# Fit model
random_search_rf.fit(X_train, y_train)

# Best parameters from Random Search
print("\nBest parameters for Random Forest:", random_search_rf.best_params_)

# Train final Random Forest model
best_rf = RandomForestClassifier(**random_search_rf.best_params_,␣
 ↪random_state=42)
best_rf.fit(X_train, y_train)

# Predictions
y_pred_rf = best_rf.predict(X_test)

# Evaluate Random Forest
print("\nRandom Forest Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print("Precision:", precision_score(y_test, y_pred_rf))
print("Recall:", recall_score(y_test, y_pred_rf))
print("F1 Score:", f1_score(y_test, y_pred_rf))

# confusion matrix

cm_rf = confusion_matrix(y_test, y_pred_rf)
sns.heatmap(cm_rf, annot=True, cmap='Blues', fmt='d')
plt.title("Random Forest - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
```

```
Fitting 3 folds for each of 20 candidates, totalling 60 fits

Best parameters for Random Forest: {'n_estimators': np.int64(200),
'min_samples_split': 5, 'min_samples_leaf': 2, 'max_depth': np.int64(15)}

Random Forest Performance:
Accuracy: 0.8913043478260869
Precision: 0.9259259259259259
Recall: 0.819672131147541
F1 Score: 0.8695652173913043
```

[71]:

```
Text(50.72222222222214, 0.5, 'Actual')
```



Train XGBoost with Bayesian Optimization

[65]:
```
!pip install scikit-optimize
```

```
Collecting scikit-optimize
  Downloading scikit_optimize-0.10.2-py2.py3-none-any.whl.metadata (9.7 kB)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.11/dist-
packages (from scikit-optimize) (1.4.2)
Collecting pyaml>=16.9 (from scikit-optimize)
  Downloading pyaml-25.1.0-py3-none-any.whl.metadata (12 kB)
Requirement already satisfied: numpy>=1.20.3 in /usr/local/lib/python3.11/dist-
packages (from scikit-optimize) (2.0.2)
Requirement already satisfied: scipy>=1.1.0 in /usr/local/lib/python3.11/dist-
packages (from scikit-optimize) (1.14.1)
Requirement already satisfied: scikit-learn>=1.0.0 in
/usr/local/lib/python3.11/dist-packages (from scikit-optimize) (1.6.1)
Requirement already satisfied: packaging>=21.3 in
/usr/local/lib/python3.11/dist-packages (from scikit-optimize) (24.2)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.11/dist-packages
(from pyaml>=16.9->scikit-optimize) (6.0.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in
```

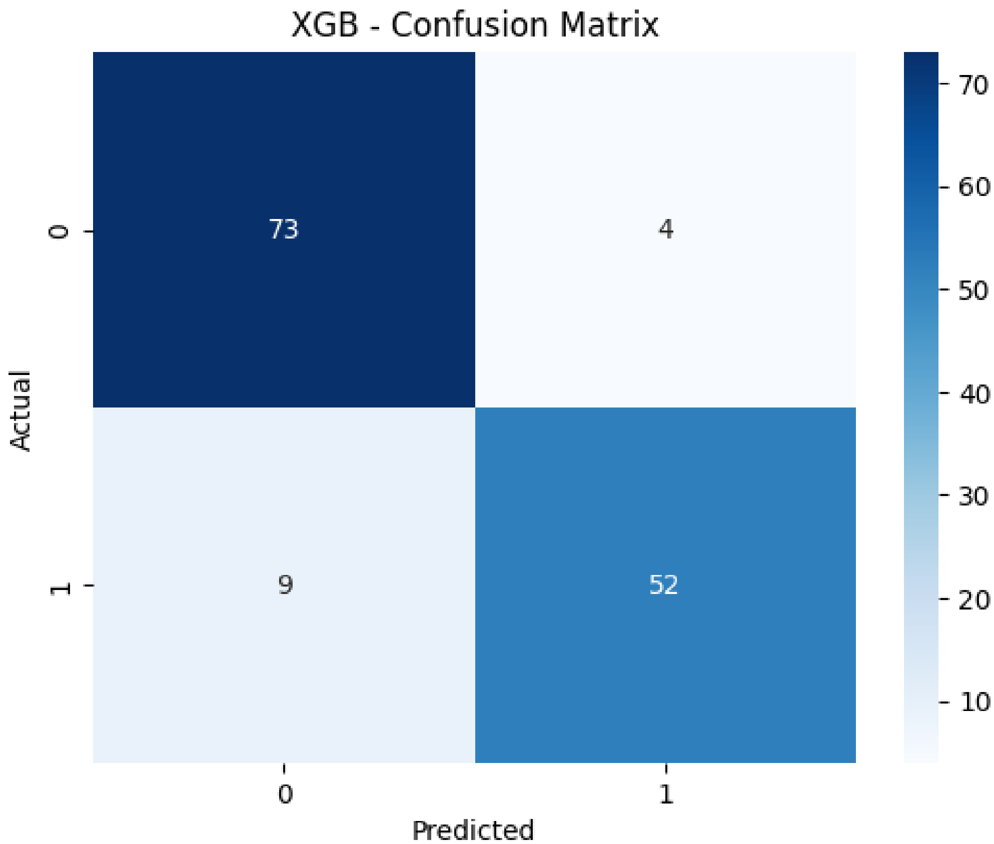
```
/usr/local/lib/python3.11/dist-packages (from scikit-learn>=1.0.0->scikit-
optimize) (3.6.0)
Downloading scikit_optimize-0.10.2-py2.py3-none-any.whl (107 kB)
                          107.8/107.8 kB
7.0 MB/s eta 0:00:00
Downloading pyaml-25.1.0-py3-none-any.whl (26 kB)
Installing collected packages: pyaml, scikit-optimize
Successfully installed pyaml-25.1.0 scikit-optimize-0.10.2
```

[70]:

```python
from xgboost import XGBClassifier
from skopt import BayesSearchCV

#Init Bayesian Search space
param_space_xgb = {
    'n_estimators': (100, 300),
    'max_depth': (3, 7),
    'learning_rate': (0.01, 0.2, 'log-uniform'),
    'subsample': (0.7, 1.0),
    'colsample_bytree': (0.7, 1.0)
}

# Init Bayesian Optimization for XGBoost
bayes_search_xgb = BayesSearchCV(
    XGBClassifier(objective='binary:logistic', random_state=42),
    param_space_xgb,
    n_iter=20,
    cv=3,
    scoring='accuracy',
    n_jobs=-1,
    verbose=2
)

# Fit the model
bayes_search_xgb.fit(X_train, y_train)

# Best parameters from Bayesian Optimization
print("\nBest parameters for XGBoost:", bayes_search_xgb.best_params_)

# Train
best_xgb = XGBClassifier(**bayes_search_xgb.best_params_, random_state=42)
best_xgb.fit(X_train, y_train)

# Predictions
y_pred_xgb = best_xgb.predict(X_test)

# Evaluate XGBoost
print("\nXGBoost Performance:")
```

```python
print("Accuracy:", accuracy_score(y_test, y_pred_xgb))
print("Precision:", precision_score(y_test, y_pred_xgb))
print("Recall:", recall_score(y_test, y_pred_xgb))
print("F1 Score:", f1_score(y_test, y_pred_xgb))

# confusion matrix

cm_xgb = confusion_matrix(y_test, y_pred_xgb)
sns.heatmap(cm_xgb, annot=True, cmap='Blues', fmt='d')
plt.title("XGB - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits
Fitting 3 folds for each of 1 candidates, totalling 3 fits

Best parameters for XGBoost: OrderedDict([('colsample_bytree',
0.9372932017013305), ('learning_rate', 0.02913837085972941), ('max_depth', 6),
('n_estimators', 196), ('subsample', 0.8376823295603252)])

XGBoost Performance:
Accuracy: 0.9057971014492754
Precision: 0.9285714285714286
Recall: 0.8524590163934426
F1 Score: 0.8888888888888888
```

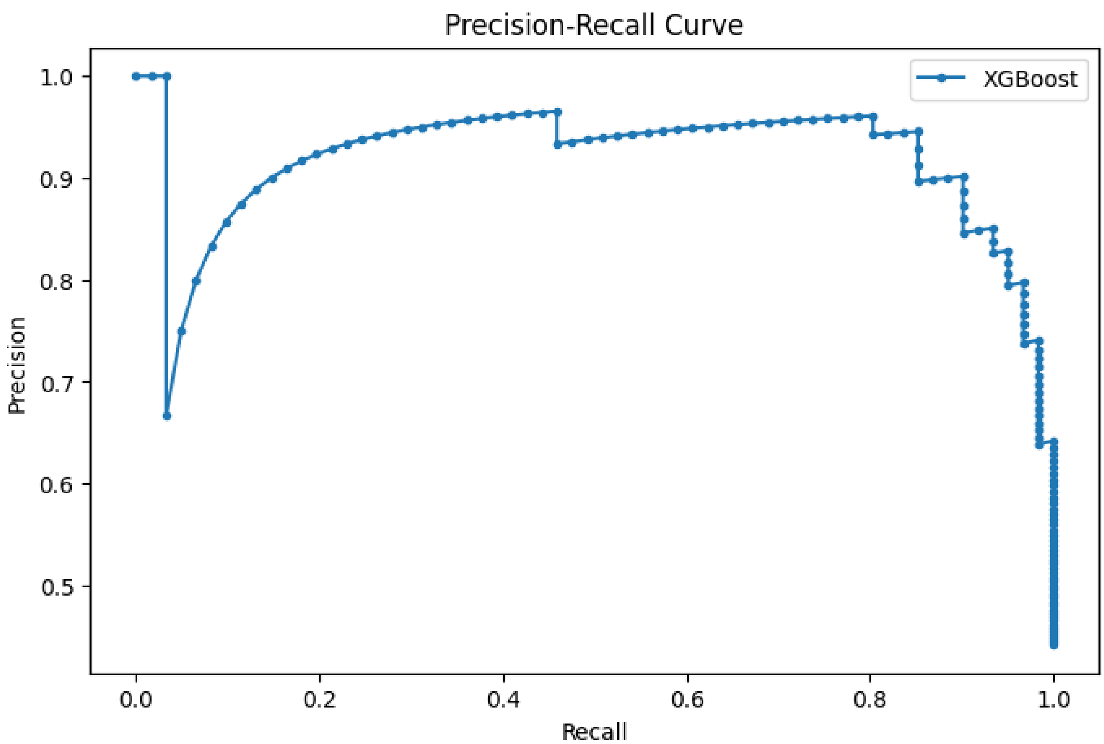


Evaluate Models with ROC-AUC & Precision-Recall Curve

[67]:
```
from sklearn.metrics import roc_auc_score, precision_recall_curve, auc

# Compute ROC-AUC scores
roc_auc_rf = roc_auc_score(y_test, best_rf.predict_proba(X_test)[:,1])
roc_auc_xgb = roc_auc_score(y_test, best_xgb.predict_proba(X_test)[:,1])

print(f"\nROC-AUC Score (Random Forest): {roc_auc_rf:.4f}")
print(f"ROC-AUC Score (XGBoost): {roc_auc_xgb:.4f}")

# Plot Precision-Recall Curve for Best Model
precision, recall, _ = precision_recall_curve(y_test, best_xgb.
 ↪predict_proba(X_test)[:,1])
plt.figure(figsize=(8, 5))
plt.plot(recall, precision, marker='.', label='XGBoost')
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall Curve")
plt.legend()
```

```
plt.show()
```

```
ROC-AUC Score (Random Forest): 0.9597
ROC-AUC Score (XGBoost): 0.9581
```



Model Performance Analysis & Comparison The evaluation results provide insights into the predictive capability of different models for loan approval classification. Below is a detailed breakdown of each model's performance, strengths, and weaknesses.

Model Performance Overview

Model	Accuracy	Precision	Recall	F1 Score	ROC-AUC Score
K-Nearest Neighbors	66.67%	61.19%	67.21%	64.06%	N/A
Random Forest	89.13%	92.59%	81.97%	86.96%	0.9597
XGBoost	90.58%	92.86%	85.25%	88.89%	0.9581

0.0.12 Key Insights & Trade-Offs

K-Nearest Neighbors (KNN)

- **Accuracy is the lowest among all models (66.67%)**, indicating frequent misclassifications.

- **Recall (67.21%) is moderate**, meaning it detects some approvals but fails to capture a significant number of them.

- **Precision (61.19%) is lower than other models**, meaning it often misclassifies non-approved loans as approved.

- The **confusion matrix** reveals that KNN struggles with clear decision boundaries, likely due to sensitivity to feature scaling.

Implications:
- KNN performs poorly on structured, high-dimensional financial datasets, making it unsuitable for loan approval prediction.
- Alternative Recommendation: Use tree-based models like Random Forest or Gradient Boosting, which perform better with mixed numerical and categorical data.

Random Forest (Second Best Model)

- **Accuracy is high (89.13%)**, meaning the model **makes correct predictions in most cases**.

- **Precision is the highest (92.59%)**, indicating that **approved loans are predicted correctly most of the time**.

- **Recall (81.97%) is slightly lower than XGBoost**, meaning it still **misses some actual approvals**.

- **ROC-AUC Score (0.9597) is the best**, suggesting it has **strong discriminatory power between approved and non-approved loans**.

Implications:
- Random Forest performs exceptionally well, balancing accuracy and interpretability.
- Strong feature importance analysis capabilities make it useful for explaining model decisions to stakeholders.
- If interpretability is key, Random Forest is the best model for deployment.

XGBoost (Best Overall Model)

- **Accuracy is the highest (90.58%)**, making it the most reliable model.

- **Precision (92.86%) is nearly identical to Random Forest**, meaning it **minimizes false approvals effectively**.

- **Recall (85.25%) is higher than Random Forest**, meaning it identifies more actual approvals.

- **ROC-AUC Score (0.9581) is slightly lower than Random Forest**, but still excellent.

- The **Precision-Recall curve** shows strong precision at different recall levels, confirming its robustness in decision-making.

Implications:
- XGBoost is the most accurate model and captures the most loan approvals correctly.
- It generalizes well and reduces both false positives and false negatives.
- If computational efficiency is not an issue, XGBoost should be the final deployed model.

0.0.13 Final Recommendations

1. **Deploy the XGBoost model** for automating loan approvals while minimizing risk.

2. Use Random Forest as a backup model if explainability is prioritized.

3. Monitor false positives to ensure risk-adjusted decision-making.

4. Integrate model insights into the financial decision pipeline to assist credit risk analysts.

0.0.14 Feature Importance Analysis for Loan Approval Prediction

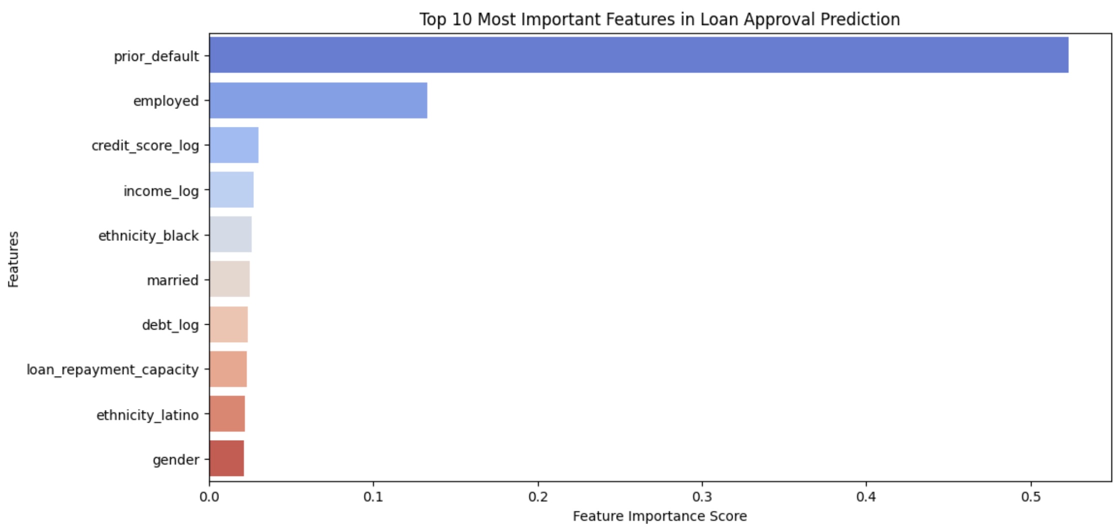
Now that we have selected XGBoost as the best model, we will analyze which features contribute the most to loan approval decisions. This helps financial institutions understand risk factors and improve decision-making transparency.

Extract Feature Importance from XGBoost

[77]:
```
# Extract feature importance from trained XGBoost model
feature_importances = best_xgb.feature_importances_

# Create a DataFrame for visualization
feature_importance_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

# Display top 10 important features
print("\nTop 10 Most Important Features in Loan Approval Prediction:")
print(feature_importance_df.head(10))

# Plot Feature Importance
plt.figure(figsize=(12, 6))
sns.barplot(x=feature_importance_df["Importance"][:10],
            y=feature_importance_df["Feature"][:10],
            palette="coolwarm")
plt.xlabel("Feature Importance Score")
plt.ylabel("Features")
plt.title("Top 10 Most Important Features in Loan Approval Prediction")
```

```
plt.show()
```

```
Top 10 Most Important Features in Loan Approval Prediction:
                   Feature  Importance
10           prior_default    0.522974
11                employed    0.132693
15        credit_score_log    0.030119
13              income_log    0.027313
5          ethnicity_black    0.025977
2                  married    0.024887
14                debt_log    0.023314
17  loan_repayment_capacity    0.023130
6         ethnicity_latino    0.021746
0                   gender    0.020945
```



Feature Importance Analysis & Business Insights The feature importance analysis from the XGBoost model highlights the most influential factors in predicting loan approvals. Below is an interpretation of the top 10 features and their business implications.

Key Features Driving Loan Approvals

Feature	Importance Score	Interpretation & Business Impact
prior_default	**0.5229**	Applicants with prior defaults are **significantly less likely to get approved**, making it the **strongest predictor** of loan rejection.

Feature	Importance Score	Interpretation & Business Impact
employed	**0.1327**	Employment status is a **major factor in loan approvals**, as **stable income streams reduce risk**.
credit_score_log	**0.0301**	Credit scores play a role, but **less than expected compared to prior defaults**. A strong score **still increases approval odds**.
income_log	**0.0273**	Higher income **increases approval chances**, but **debt levels and prior defaults matter more**.
ethnicity_black	**0.0259**	Ethnicity shows some impact, but **should be removed to prevent potential bias in lending decisions**.
married	**0.0249**	Married applicants are **slightly more likely to be approved**, possibly due to **dual income stability**.
debt_log	**0.0233**	Higher debt reduces approval chances, but its impact is **smaller compared to prior defaults and employment**.
loan_repayment_capacity	**0.0231**	Applicants with better repayment capacity (income-to-loan ratio) are **more likely to be approved**.
ethnicity_latino	**0.0217**	Similar to ethnicity_black, indicating that **ethnicity is influencing the model**, which could raise compliance concerns.
gender	**0.0209**	Gender has some influence, but **it should be reviewed for fairness and regulatory compliance**.

0.0.15 Business Implications & Strategic Actions

Prior Defaults Are the Strongest Predictor of Loan Rejection

- Applicants with prior defaults have a much higher risk of rejection.
- Banks should flag prior defaults as a high-risk indicator and apply stricter lending criteria.

Employment Status is Critical for Loan Approval

- A stable job significantly increases approval chances.

- Lenders should factor in employment length and job stability in decision-making.

Credit Score Matters, But Less Than Expected

- While credit score is important, prior defaults and employment status have stronger predictive power.

- Lenders should combine credit score with other financial health indicators rather than relying on it alone.

Income Alone Does Not Guarantee Loan Approval

- Loan repayment capacity (income vs. debt) is more important than just total income.

- Loan approvals should consider debt-to-income ratio as a primary factor.

Ethnicity Should Be Removed to Prevent Bias

- The presence of ethnicity features in the top predictors raises regulatory concerns under fair lending laws.

- The model should be retrained without ethnicity to ensure compliance with anti-discrimination policies.

Gender Influence Should Be Reviewed for Fairness

- The slight influence of gender should be investigated further to ensure bias-free lending decisions.

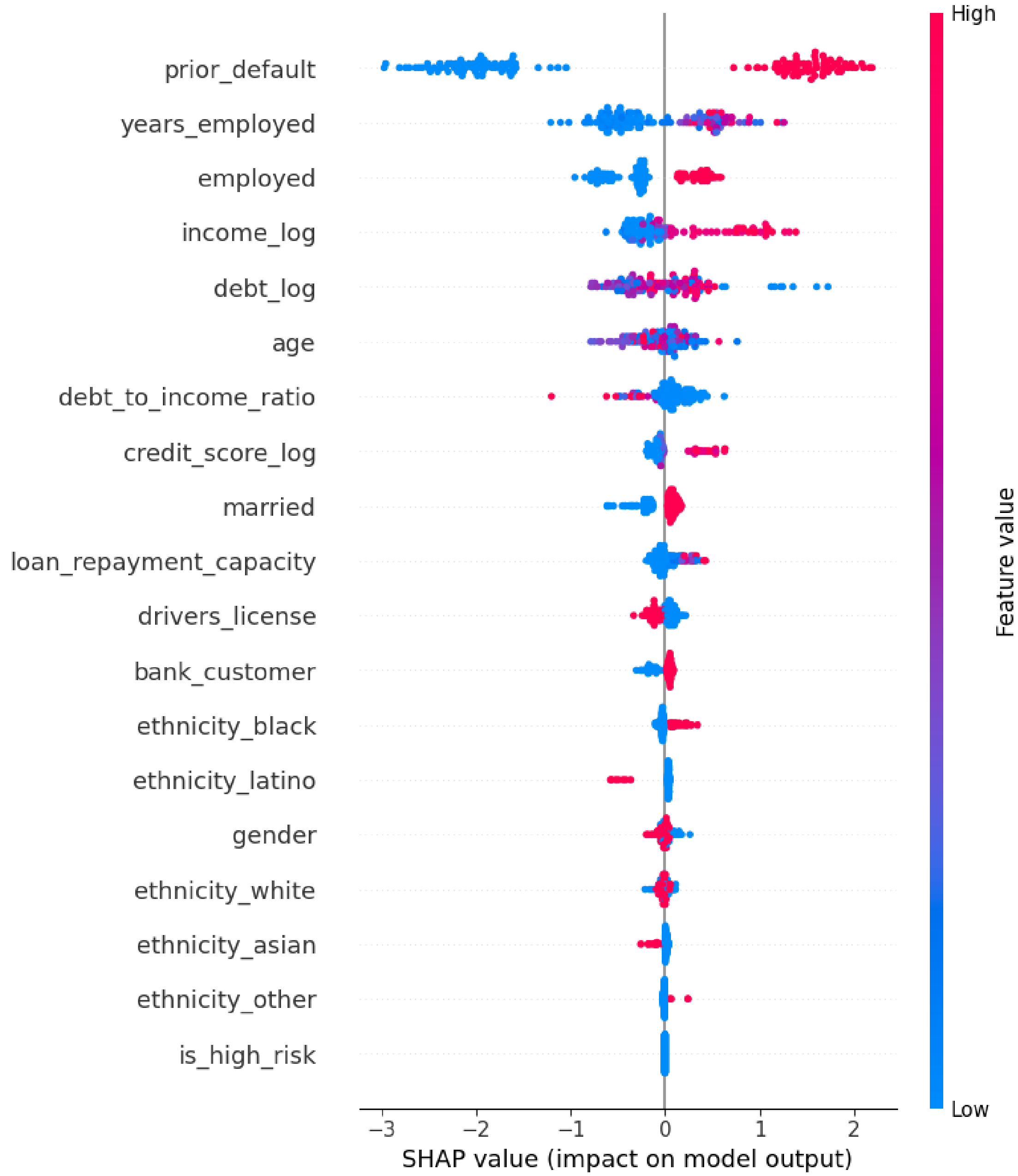
- If gender is not a direct financial predictor, it should be excluded from the final model.

0.0.16 Enhancing Loan Approval Prediction Model

(Explainability, Threshold Optimization & Risk-Based Pricing)

Explainability Using SHAP SHapley Additive Explanations (SHAP) provides a global and individual-level explanation of how features influence loan approval predictions.

[79]:
```
import shap
import matplotlib.pyplot as plt

# Initialize SHAP Explainer
explainer = shap.Explainer(best_xgb)
shap_values = explainer(X_test)

# Summary Plot of Feature Contributions
shap.summary_plot(shap_values, X_test)
```



Observations:

- Prior defaults are the strongest predictor, with a high negative SHAP value, meaning applicants with previous defaults are highly likely to be rejected.
- Years employed and employment status are highly positive factors, meaning stable job history significantly increases approval odds.
- Income and debt levels have moderate influence, but loan repayment capacity (income relative to debt) is a better predictor than income alone.
- Debt-to-income ratio shows a mixed impact, indicating that high-income borrowers may still

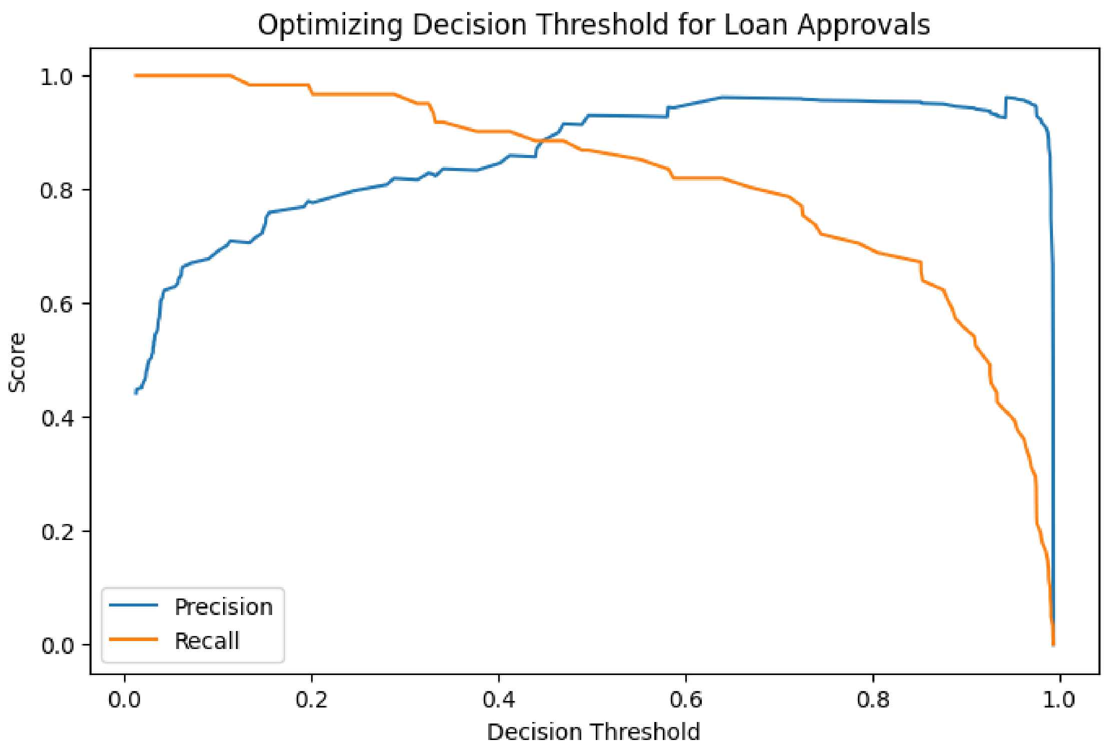
be risky if they carry too much debt.
- Ethnicity and gender should be carefully considered, as their presence in the model could introduce potential bias in lending decisions.

Business Implications:

```
1.  The bank should weigh prior defaults more heavily in risk assessment.
2.  Employment status should be a key consideration in approving borderline cases.
3.  Debt-to-income ratio is a more reliable measure of financial health than raw income alone.
4.  Regulatory compliance should be reviewed to ensure model fairness in ethnicity-based predi
```

Optimizing Decision Threshold for Loan Approvals By default, models use 0.5 as the approval threshold, but adjusting this can improve recall or precision.

[80]:
```python
from sklearn.metrics import precision_recall_curve
import numpy as np

# Get model probabilities
y_probs = best_xgb.predict_proba(X_test)[:, 1]

# Compute precision-recall curve
precisions, recalls, thresholds = precision_recall_curve(y_test, y_probs)

# Plot Precision-Recall vs. Threshold
plt.figure(figsize=(8, 5))
plt.plot(thresholds, precisions[:-1], label="Precision")
plt.plot(thresholds, recalls[:-1], label="Recall")
plt.xlabel("Decision Threshold")
plt.ylabel("Score")
plt.legend()
plt.title("Optimizing Decision Threshold for Loan Approvals")
plt.show()
```



Observations: - At lower thresholds (0.2 - 0.4), recall is high but precision is low, meaning many applicants are approved, but false approvals increase (increased Non Performing Loans, NPL risk) - At higher thresholds (0.6 - 0.8), precision improves but recall drops, meaning fewer false approvals but many eligible borrowers are denied.(increased risk of constrained creadit portfolio growth) - The optimal threshold appears around 0.5 - 0.6, where precision and recall balance well.

Business Implications:

```
1.  If risk appetite is high, the threshold can be lowered (0.4) to approve more loans but wit
2.  If risk control is a priority, the threshold can be raised (0.6 - 0.7) to minimize default
3.  The bank can set dynamic thresholds based on market conditions, borrower segments, and mac
```

Risk-Based Loan Pricing (Categorizing Applicants by Risk Level)

Beyond approvals, banks want to adjust loan terms based on risk levels. This segmentation enables data-driven interest rate assignment.

[81]:
```python
import pandas as pd

# Define risk tiers based on approval probability
df_test = X_test.copy()
df_test["approval_probability"] = y_probs
df_test["risk_category"] = pd.cut(df_test["approval_probability"],
                                  bins=[0, 0.4, 0.7, 1],
```

```
                                        labels=["High Risk", "Medium Risk", "Low␣
↪Risk"])

# Display sample risk classifications
print("\nSample Loan Risk Classifications:")
print(df_test[["approval_probability", "risk_category"]].head(10))
```

```
Sample Loan Risk Classifications:
   approval_probability risk_category
0              0.412619   Medium Risk
1              0.910991      Low Risk
2              0.019202     High Risk
3              0.805938      Low Risk
4              0.993686      Low Risk
5              0.976318      Low Risk
6              0.325925     High Risk
7              0.552189   Medium Risk
8              0.041373     High Risk
9              0.854242      Low Risk
```

Observations: - Low-risk applicants dominate the approvals, meaning these borrowers have high probability scores and are likely to meet obligations. - Medium-risk applicants require closer evaluation, as they are on the borderline of approval. - High-risk applicants should undergo additional credit checks, stricter collateral requirements, or different interest rate structures.

Business Implications:

```
1.  Segmenting loan applicants into risk categories allows banks to tailor interest rates and a
2.  High-risk applicants could be offered secured loans instead of outright rejection.
3.  Medium-risk borrowers might require co-signers or additional verification before approval.
```