# LOAN APPROVAL PREDICTOR

## Business challenge.

Financial institutions struggle with evaluating loan applicants efficiently, leading to approval delays, high default rates and potential biases. Traditional models rely on rigid criteria, overlooking valuable insights from alternative data.

Loan Approval Predictor aims to build a predictive model that leverages historical loan data, borrower profiles and classification techniques to refine loan approval decisions. By enhancing risk assessment and promoting fair, accessible credit opportunities, it seeks to improve financial inclusion and operational efficiency.

## Objectives

1. Enhance loan approval accuracy by leveraging historical data and predictive modeling techniques.
2. Minimize default risk by identifying high-risk applicants through classification algorithms.
3. Promote fairness in lending by analyzing biases in loan approval decisions and improving accessibility.
4. Streamline decision-making by automating loan approval predictions to reduce processing time.
5. Improve model interpretability to ensure stakeholders understand key factors influencing loan approvals.

### Analysis Workflow

```
A[Data Loading] --> B[Exploratory Data Analysis]
B --> C[Data Preprocessing]
C --> D[Model Building]
D --> E[Model Evaluation]
E --> F[Feature Importance]
F --> G[Conclusions & Recommendations]
```

In [51]:
```
#importing necessary libraries
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, cross_validat
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,precision_score, recall_score, f1_score, pre
from sklearn.ensemble import RandomForestClassifier
```

```
from imblearn.over_sampling import SMOTE
import joblib
```

# 1.Loading and Inspecting data

In [3]:
```
#Load the dataset in Python using pandas and inspect the first few rows

df=pd.read_csv("loan_data.csv")
df.head(10)
```

Out[3]:

| | person_age | person_gender | person_education | person_income | person_emp_exp | person_home_ownersl |
|---|---|---|---|---|---|---|
| **0** | 22.0 | female | Master | 71948.0 | 0 | RE |
| **1** | 21.0 | female | High School | 12282.0 | 0 | OV |
| **2** | 25.0 | female | High School | 12438.0 | 3 | MORTGA |
| **3** | 23.0 | female | Bachelor | 79753.0 | 0 | RE |
| **4** | 24.0 | male | Master | 66135.0 | 1 | RE |
| **5** | 21.0 | female | High School | 12951.0 | 0 | OV |
| **6** | 26.0 | female | Bachelor | 93471.0 | 1 | RE |
| **7** | 24.0 | female | High School | 95550.0 | 5 | RE |
| **8** | 24.0 | female | Associate | 100684.0 | 3 | RE |
| **9** | 21.0 | female | High School | 12739.0 | 0 | OV |

This dataset contains loan application records, including applicant demographics, financial details, credit history and loan approval status. It aims to identify patterns in borrower profiles and factors influencing loan approvals.

In [4]:
```
# Checking the shape of the dataset (rows, columns)

df.shape
```

Out[4]:
```
(45000, 14)
```

In [5]:
```
# Checking dataset structure and column details

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45000 entries, 0 to 44999
Data columns (total 14 columns):
 #   Column                      Non-Null Count  Dtype  
---  ------                      --------------  -----  
 0   person_age                  45000 non-null  float64
 1   person_gender               45000 non-null  object 
 2   person_education            45000 non-null  object 
 3   person_income               45000 non-null  float64
 4   person_emp_exp              45000 non-null  int64  
 5   person_home_ownership       45000 non-null  object 
 6   loan_amnt                   45000 non-null  float64
 7   loan_intent                 45000 non-null  object 
 8   loan_int_rate               45000 non-null  float64
```

```
 9   loan_percent_income             45000 non-null  float64
 10  cb_person_cred_hist_length      45000 non-null  float64
 11  credit_score                    45000 non-null  int64
 12  previous_loan_defaults_on_file  45000 non-null  object
 13  loan_status                     45000 non-null  int64
dtypes: float64(6), int64(3), object(5)
memory usage: 4.8+ MB
```

The above displays the number of non-null values, data types and memory usage, helping us identify missing data and potential type conversions.

In [6]:
```
# Convert categorical variables to category type
categorical_cols = ['person_gender', 'person_education', 'person_home_ownership',
                    'loan_intent', 'previous_loan_defaults_on_file', 'loan_status']
for col in categorical_cols:
    df[col] = df[col].astype('category')
categorical_cols
```

Out[6]:
```
['person_gender',
 'person_education',
 'person_home_ownership',
 'loan_intent',
 'previous_loan_defaults_on_file',
 'loan_status']
```

In [7]:
```
# Handle outliers using IQR
numeric_cols = ['person_age', 'person_income', 'person_emp_exp', 'loan_amnt',
                'loan_int_rate', 'loan_percent_income', 'cb_person_cred_hist_length',
                'credit_score']
for col in numeric_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df[col] = df[col].clip(lower=lower_bound, upper=upper_bound)
numeric_cols
```

Out[7]:
```
['person_age',
 'person_income',
 'person_emp_exp',
 'loan_amnt',
 'loan_int_rate',
 'loan_percent_income',
 'cb_person_cred_hist_length',
 'credit_score']
```

In [8]:
```
# One-hot encode categorical variables (excluding loan_status as it's the target)
categorical_cols_to_encode = ['person_gender', 'person_education', 'person_home_ownersh
                              'loan_intent', 'previous_loan_defaults_on_file']

# Apply one-hot encoding
df_encoded = pd.get_dummies(df, columns=categorical_cols_to_encode, drop_first=True)

# Ensure loan_status is numeric
df_encoded['loan_status'] = df_encoded['loan_status'].astype(int)

print(df_encoded.head())  # View the first few rows
print(df_encoded.columns)  # Check the column names
```

```
   person_age  person_income  person_emp_exp  loan_amnt  loan_int_rate  \
0        22.0        71948.0             0.0  23093.125          16.02
```

```
1        21.0        12282.0             0.0   1000.000          11.14
2        25.0        12438.0             3.0   5500.000          12.87
3        23.0        79753.0             0.0  23093.125          15.23
4        24.0        66135.0             1.0  23093.125          14.27

   loan_percent_income  cb_person_cred_hist_length  credit_score  loan_status  \
0                 0.37                         3.0         561.0            1
1                 0.08                         2.0         504.0            0
2                 0.37                         3.0         635.0            1
3                 0.37                         2.0         675.0            1
4                 0.37                         4.0         586.0            1

   person_gender_male  ...  person_education_Master  \
0                   0  ...                        1
1                   0  ...                        0
2                   0  ...                        0
3                   0  ...                        0
4                   1  ...                        1

   person_home_ownership_OTHER  person_home_ownership_OWN  \
0                            0                          0
1                            0                          1
2                            0                          0
3                            0                          0
4                            0                          0

   person_home_ownership_RENT  loan_intent_EDUCATION  \
0                           1                      0
1                           0                      1
2                           0                      0
3                           1                      0
4                           1                      0

   loan_intent_HOMEIMPROVEMENT  loan_intent_MEDICAL  loan_intent_PERSONAL  \
0                            0                    0                     1
1                            0                    0                     0
2                            0                    1                     0
3                            0                    1                     0
4                            0                    1                     0

   loan_intent_VENTURE  previous_loan_defaults_on_file_Yes
0                    0                                   0
1                    0                                   1
2                    0                                   0
3                    0                                   0
4                    0                                   0

[5 rows x 23 columns]
Index(['person_age', 'person_income', 'person_emp_exp', 'loan_amnt',
       'loan_int_rate', 'loan_percent_income', 'cb_person_cred_hist_length',
       'credit_score', 'loan_status', 'person_gender_male',
       'person_education_Bachelor', 'person_education_Doctorate',
       'person_education_High School', 'person_education_Master',
       'person_home_ownership_OTHER', 'person_home_ownership_OWN',
       'person_home_ownership_RENT', 'loan_intent_EDUCATION',
       'loan_intent_HOMEIMPROVEMENT', 'loan_intent_MEDICAL',
       'loan_intent_PERSONAL', 'loan_intent_VENTURE',
       'previous_loan_defaults_on_file_Yes'],
      dtype='object')
```

In [9]:

```
# Assuming df_encoded is the DataFrame after one-hot encoding
# Separate numerical and categorical/binary columns
numerical_cols = ['person_age', 'person_income', 'person_emp_exp', 'loan_amnt',
                  'loan_int_rate', 'loan_percent_income', 'cb_person_cred_hist_length',
```

```
                  'credit_score']
binary_cols = [col for col in df_encoded.columns if col not in numerical_cols and col !
```

In [10]:
```
# Summary Statistics for Numerical Columns
print("Summary Statistics for Numerical Columns:")
numerical_summary = df_encoded[numerical_cols].describe()
print(numerical_summary)
```

```
Summary Statistics for Numerical Columns:
         person_age  person_income  person_emp_exp     loan_amnt  \
count  45000.000000   45000.000000    45000.000000  45000.000000
mean      27.444733   75677.400428        5.170311   9411.042900
std        4.930055   38071.779506        5.137068   5832.950765
min       20.000000    8000.000000        0.000000    500.000000
25%       24.000000   47204.000000        1.000000   5000.000000
50%       26.000000   67048.000000        4.000000   8000.000000
75%       30.000000   95789.250000        8.000000  12237.250000
max       39.000000  168667.125000       18.500000  23093.125000

       loan_int_rate  loan_percent_income  cb_person_cred_hist_length  \
count   45000.000000         45000.000000                45000.000000
mean       11.005676             0.138842                    5.783711
std         2.976069             0.084360                    3.578787
min         5.420000             0.000000                    2.000000
25%         8.590000             0.070000                    3.000000
50%        11.010000             0.120000                    4.000000
75%        12.990000             0.190000                    8.000000
max        19.590000             0.370000                   15.500000

       credit_score
count   45000.00000
mean      632.80830
std        49.80135
min       497.50000
25%       601.00000
50%       640.00000
75%       670.00000
max       773.50000
```

In [11]:
```
#Summary Statistics for Binary (One-Hot Encoded) Columns
print("\nSummary Statistics for Binary Columns (Proportion of 1s):")
binary_summary = df_encoded[binary_cols].mean().sort_values(ascending=False)
print(binary_summary)
```

```
Summary Statistics for Binary Columns (Proportion of 1s):
person_gender_male                    0.552022
person_home_ownership_RENT            0.520956
previous_loan_defaults_on_file_Yes    0.507956
person_education_Bachelor             0.297756
person_education_High School          0.266044
loan_intent_EDUCATION                 0.203400
loan_intent_MEDICAL                   0.189956
loan_intent_VENTURE                   0.173756
loan_intent_PERSONAL                  0.167822
person_education_Master               0.155111
loan_intent_HOMEIMPROVEMENT           0.106289
person_home_ownership_OWN             0.065578
person_education_Doctorate            0.013800
person_home_ownership_OTHER           0.002600
dtype: float64
```

Key Observations

- Most applicants are male (55.2%) and renters (52.1%).

- Previous loan defaults are roughly balanced (50.8% Yes, 49.2% No).
- Education-related loans make up only 20.3% of all loans.
- Doctorate degrees (1.38%) and 'Other' home ownership types (0.26%) are rare.

In [12]:
```
#Summary Statistics for Target Variable (loan_status)
print("\nSummary Statistics for Target Variable (loan_status):")
df_encoded['loan_status'].value_counts(normalize=True)
```

```
Summary Statistics for Target Variable (loan_status):
```

Out[12]:
```
0    0.777778
1    0.222222
Name: loan_status, dtype: float64
```

In [13]:
```
df_encoded.to_csv("cleaned_loan_data.csv", index=False)
print("Cleaned data saved to 'cleaned_loan_data.csv'")
```

```
Cleaned data saved to 'cleaned_loan_data.csv'
```

## 2. Exploratory Data Analysis(EDA)

In [14]:
```
# Load the cleaned dataset
df_encoded= pd.read_csv("cleaned_loan_data.csv")
df_encoded
```

Out[14]:

| | person_age | person_income | person_emp_exp | loan_amnt | loan_int_rate | loan_percent_income | cb_ |
|---|---|---|---|---|---|---|---|
| **0** | 22.0 | 71948.0 | 0.0 | 23093.125 | 16.02 | 0.37 | |
| **1** | 21.0 | 12282.0 | 0.0 | 1000.000 | 11.14 | 0.08 | |
| **2** | 25.0 | 12438.0 | 3.0 | 5500.000 | 12.87 | 0.37 | |
| **3** | 23.0 | 79753.0 | 0.0 | 23093.125 | 15.23 | 0.37 | |
| **4** | 24.0 | 66135.0 | 1.0 | 23093.125 | 14.27 | 0.37 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **44995** | 27.0 | 47971.0 | 6.0 | 15000.000 | 15.66 | 0.31 | |
| **44996** | 37.0 | 65800.0 | 17.0 | 9000.000 | 14.07 | 0.14 | |
| **44997** | 33.0 | 56942.0 | 7.0 | 2771.000 | 10.02 | 0.05 | |
| **44998** | 29.0 | 33164.0 | 4.0 | 12000.000 | 13.23 | 0.36 | |
| **44999** | 24.0 | 51609.0 | 1.0 | 6665.000 | 17.05 | 0.13 | |

45000 rows × 23 columns

In [15]:
```
# Summary statistics
print("EDA Summary Statistics:")
df_encoded.describe(include='all')
```

```
EDA Summary Statistics:
```

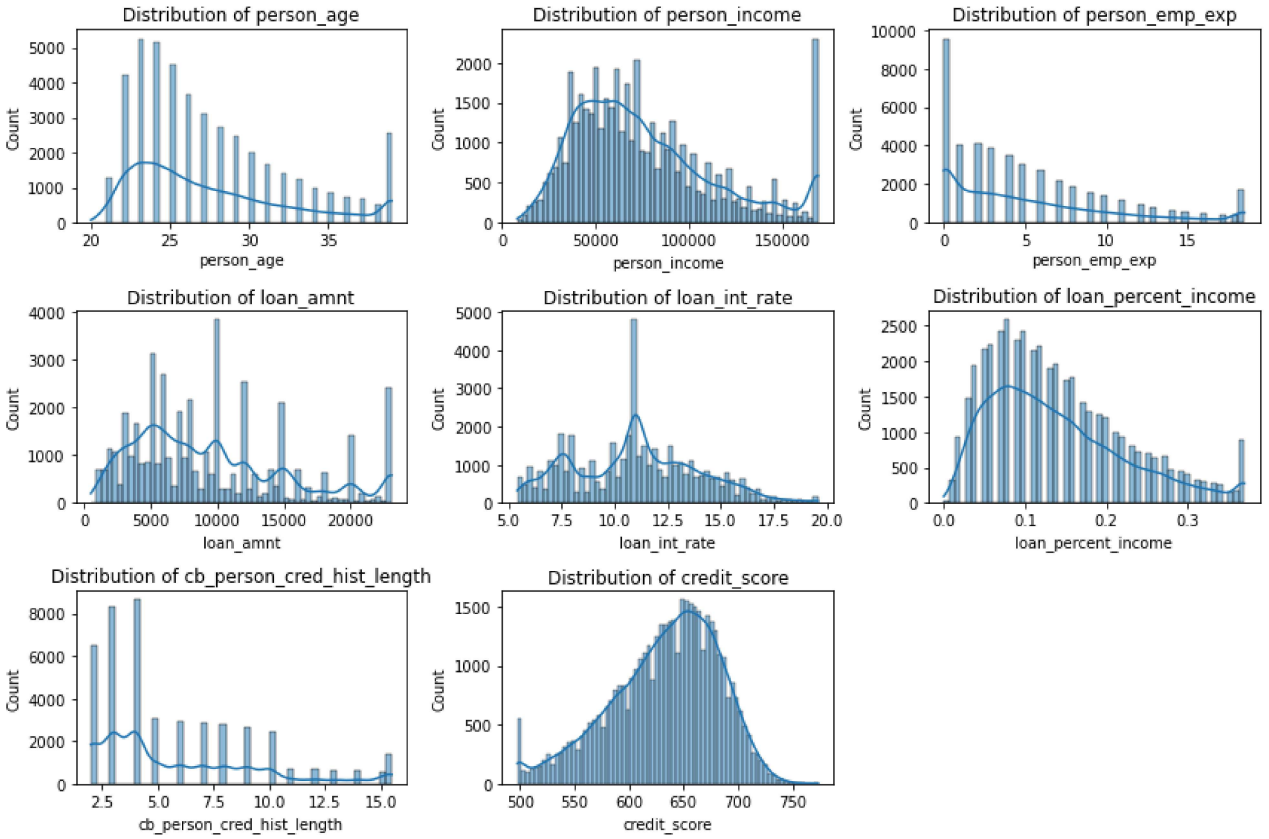
Out[15]:

| | person_age | person_income | person_emp_exp | loan_amnt | loan_int_rate | loan_percent_income |
|---|---|---|---|---|---|---|
| **count** | 45000.000000 | 45000.000000 | 45000.000000 | 45000.000000 | 45000.000000 | 45000.000000 |
| **mean** | 27.444733 | 75677.400428 | 5.170311 | 9411.042900 | 11.005676 | 0.138842 |

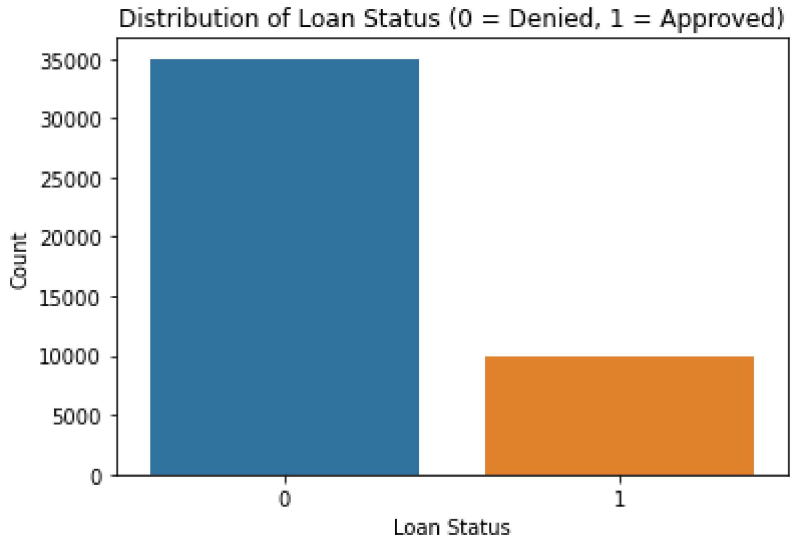
| | person_age | person_income | person_emp_exp | loan_amnt | loan_int_rate | loan_percent_income |
|---|---|---|---|---|---|---|
| **std** | 4.930055 | 38071.779506 | 5.137068 | 5832.950765 | 2.976069 | 0.084360 |
| **min** | 20.000000 | 8000.000000 | 0.000000 | 500.000000 | 5.420000 | 0.000000 |
| **25%** | 24.000000 | 47204.000000 | 1.000000 | 5000.000000 | 8.590000 | 0.070000 |
| **50%** | 26.000000 | 67048.000000 | 4.000000 | 8000.000000 | 11.010000 | 0.120000 |
| **75%** | 30.000000 | 95789.250000 | 8.000000 | 12237.250000 | 12.990000 | 0.190000 |
| **max** | 39.000000 | 168667.125000 | 18.500000 | 23093.125000 | 19.590000 | 0.370000 |

8 rows × 23 columns

In [16]:
```
#Visualize Distribution of Key Numerical Columns
plt.figure(figsize=(12, 8))
for i, col in enumerate(numerical_cols, 1):
    plt.subplot(3, 3, i)
    sns.histplot(df_encoded[col], kde=True)
    plt.title(f'Distribution of {col}')
    plt.tight_layout()
plt.show()
```

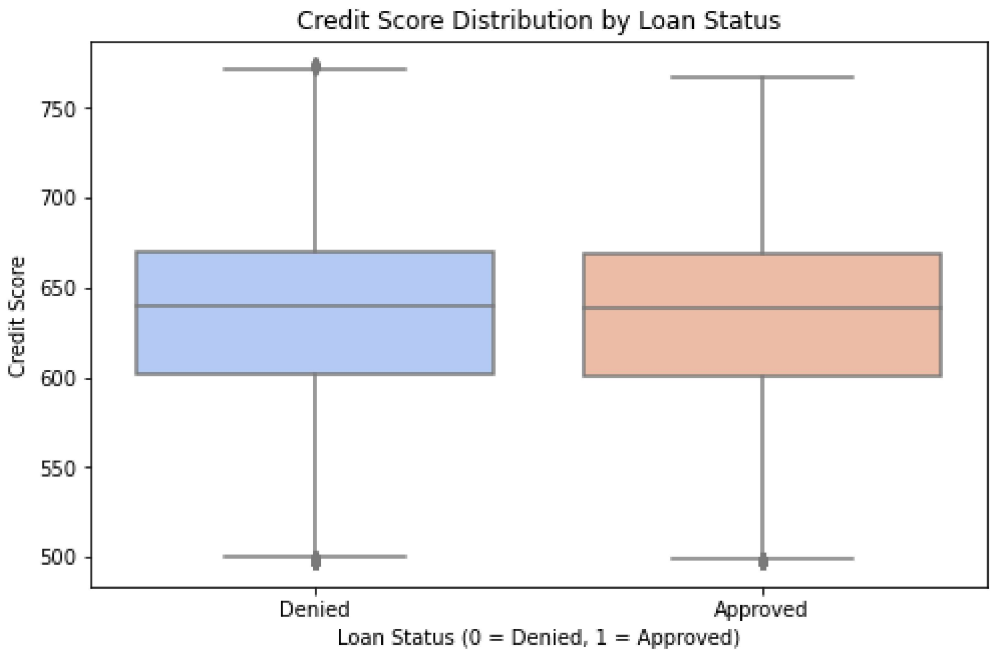



In [17]:
```
#Visualize Distribution of loan_status
plt.figure(figsize=(6, 4))
sns.countplot(x='loan_status', data=df_encoded)
plt.title('Distribution of Loan Status (0 = Denied, 1 = Approved)')
plt.xlabel('Loan Status')
plt.ylabel('Count')
plt.show()
```

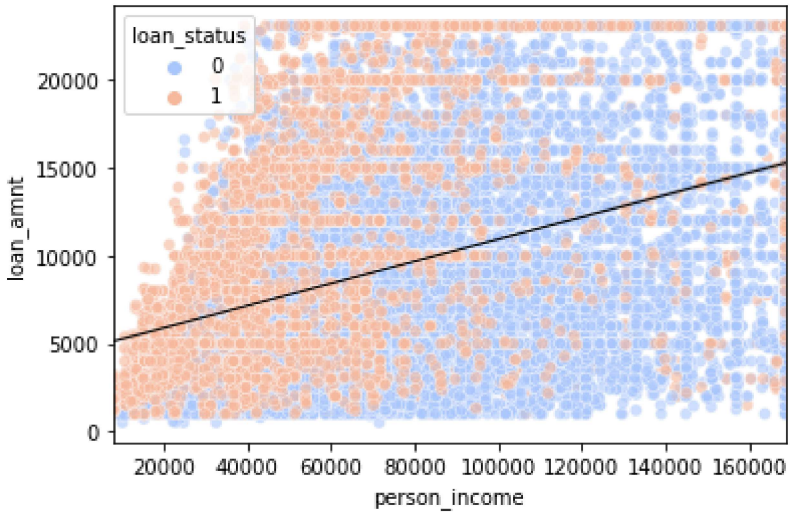



This indicates that a substantial majority of loan applications in the dataset were rejected, highlighting a potential class imbalance that could impact predictive modeling efforts for loan approval decisions.

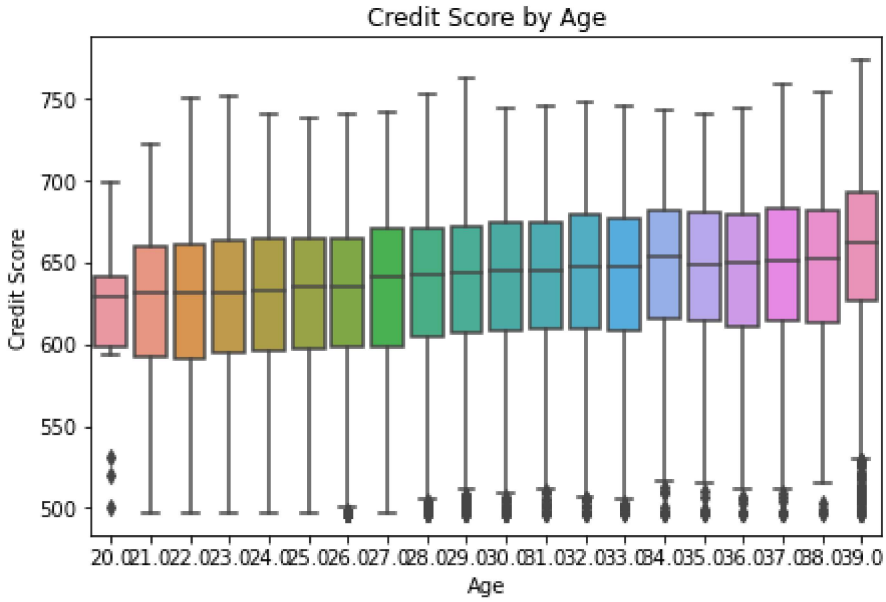
In [18]:
```
# Box plot of credit_score by loan_status
plt.figure(figsize=(8, 5))
sns.boxplot(data=df_encoded, x='loan_status', y='credit_score', palette='coolwarm')
plt.title('Credit Score Distribution by Loan Status')
plt.xlabel('Loan Status (0 = Denied, 1 = Approved)')
plt.ylabel('Credit Score')
plt.xticks(ticks=[0, 1], labels=['Denied', 'Approved'])
plt.show()
```




In [19]:
```
# Income vs Loan Amount
sns.scatterplot(data=df, x='person_income', y='loan_amnt', hue='loan_status', palette='
sns.regplot(data=df, x='person_income', y='loan_amnt', scatter=False, color='black', li
correlation = df['person_income'].corr(df['loan_amnt'])
```

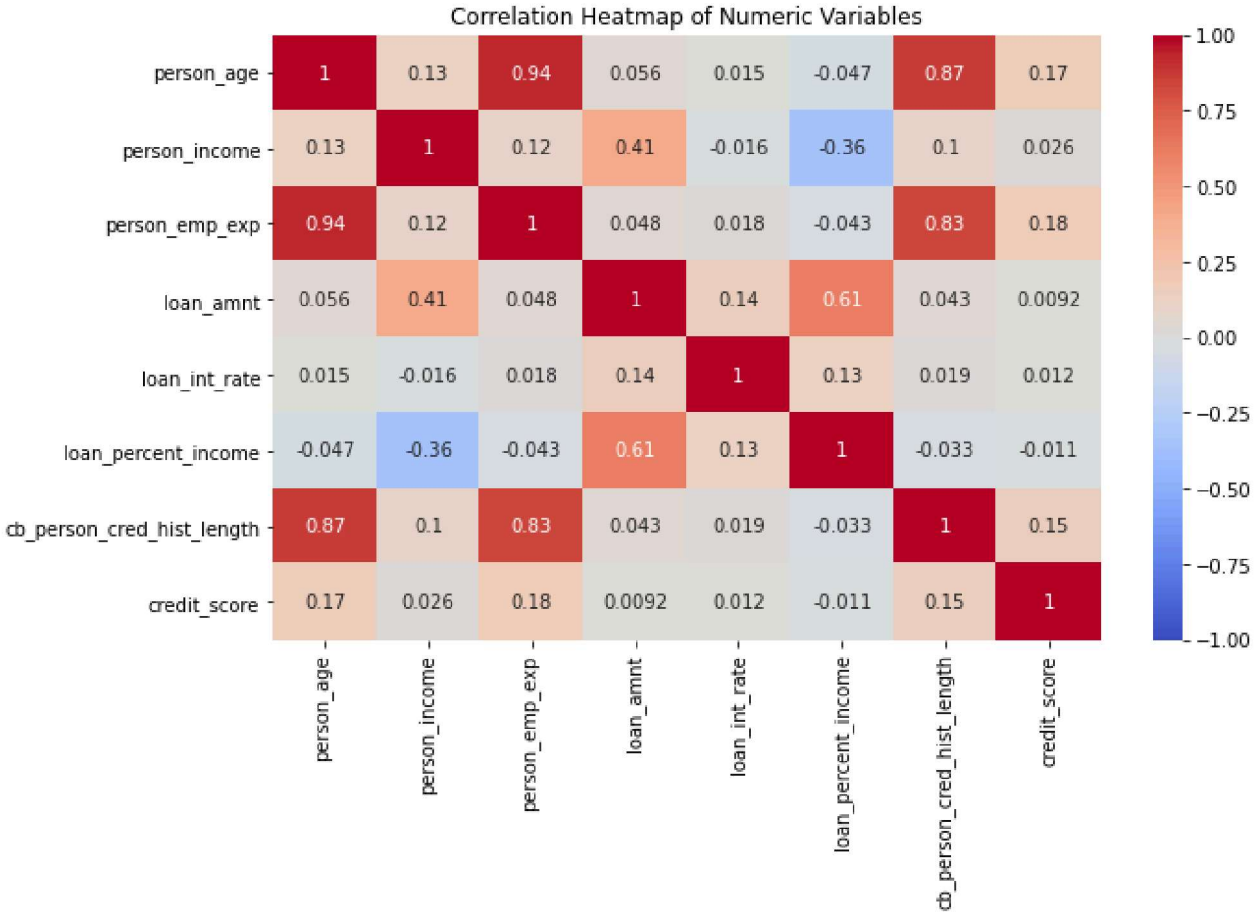



In [20]:
```
# Credit Score by Age (Box Plot)
plt.figure(figsize=(12, 8))
plt.subplot(2, 2, 1)
sns.boxplot(x='person_age', y='credit_score', data=df)
plt.title('Credit Score by Age')
plt.xlabel('Age')
plt.ylabel('Credit Score')
plt.tight_layout()
plt.show()
```




Credit scores tend to improve and stabilize with age, with younger people showing more variation and lower scores, while older groups have higher, more consistent scores.

In [21]:
```
# Correlation heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(df[numeric_cols].corr(), annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Heatmap of Numeric Variables')
plt.show()
```




- person_age and cb_person_cred_hist_length: 0.87 (strong positive correlation), meaning older people tend to have longer credit histories.
- person_age and person_emp_exp: 0.94 (very strong positive correlation), indicating older individuals generally have more employment experience.
- person_income and loan_amnt: 0.41 (moderate positive correlation), suggesting higher income is associated with larger loan amounts.
- loan_amnt and loan_percent_income: 0.61 (moderate positive correlation), meaning larger loans are a bigger percentage of income.
- person_income and loan_percent_income: -0.36 (moderate negative correlation), indicating higher income is linked to a smaller loan-to-income ratio.
- person_age and loan_percent_income: -0.47 (moderate negative correlation), suggesting older people have a smaller loan-to-income ratio.
- Most other pairs (e.g., credit_score with most variables) show weak correlations (close to 0), indicating little relationship.

Summary: Age strongly correlates with employment experience and credit history length. Loan amount and loan-to-income ratio are moderately related, and higher income tends to lower the loan-to-income ratio. Credit score has weak correlations with most variables.

In [22]:
```
# Insight
correlation = df['person_income'].corr(df['loan_amnt'])
print(f"\nInsight: There is a correlation of {correlation:.2f} between person_income an
```

Insight: There is a correlation of 0.41 between person_income and loan_amnt, suggesting higher incomes are associated with larger loans.

# 3.Modelling

In [ ]:
```
# Define features (X) and target (y)

X = df_encoded.drop('loan_status', axis=1)
y = df_encoded['loan_status'].astype(int)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4
```

In [61]:
```
# Class imbalance
print(y_train.value_counts())
```

```
0    28010
1     7990
Name: loan_status, dtype: int64
```

In [62]:
```
# Initialize SMOTE
smote = SMOTE(sampling_strategy='auto', random_state=42)

# Apply oversampling to training data
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Check new class distribution
print(y_train_resampled.value_counts())
```

```
1    28010
0    28010
Name: loan_status, dtype: int64
```

In [ ]:
```
model = LogisticRegression(max_iter=500)
model.fit(X_train, y_train)
# Perform cross-validation
cv_scores = cross_val_score(model, X_train, y_train, cv=5, scoring='accuracy')

# Make predictions
train_pred = model.predict(X_train)
test_pred = model.predict(X_test)


# Evaluate accuracy
train_accuracy = accuracy_score(y_train, train_pred)
test_accuracy = accuracy_score(y_test, test_pred)

# Print results
print(f"Train Accuracy: {train_accuracy:.3f}")
print(f"Test Accuracy: {test_accuracy:.3f}")
print(f"Mean CV Accuracy: {cv_scores.mean():.3f}")
print(f"Standard Deviation of CV: {cv_scores.std():.3f}")
```

```
Train Accuracy: 0.886
Test Accuracy: 0.884
Mean CV Accuracy: 0.833
Standard Deviation of CV: 0.029
```

In [47]:
```
print(classification_report(y_test, y_pred))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.92 | 0.93 | 0.93 | 6990 |
| 1 | 0.75 | 0.72 | 0.74 | 2010 |
| accuracy | | | 0.88 | 9000 |
| macro avg | 0.83 | 0.83 | 0.83 | 9000 |
| weighted avg | 0.88 | 0.88 | 0.88 | 9000 |

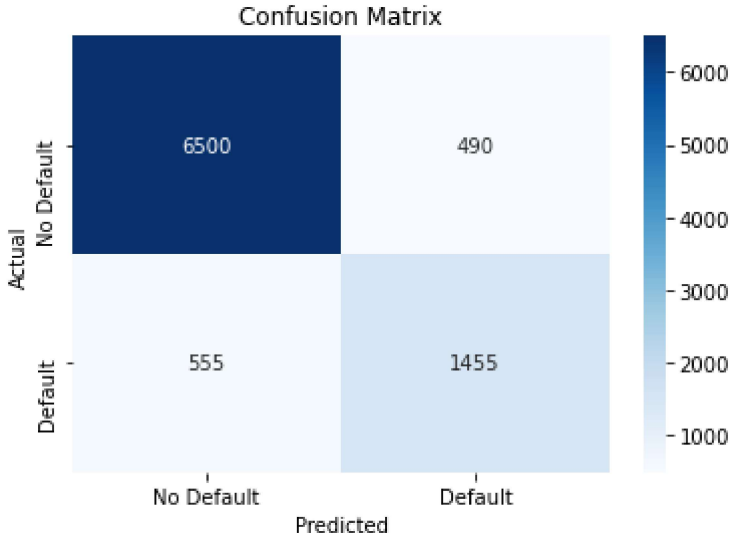
## Train the Logistic Regression Model

In [39]:
```
model = LogisticRegression(class_weight='balanced', random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

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

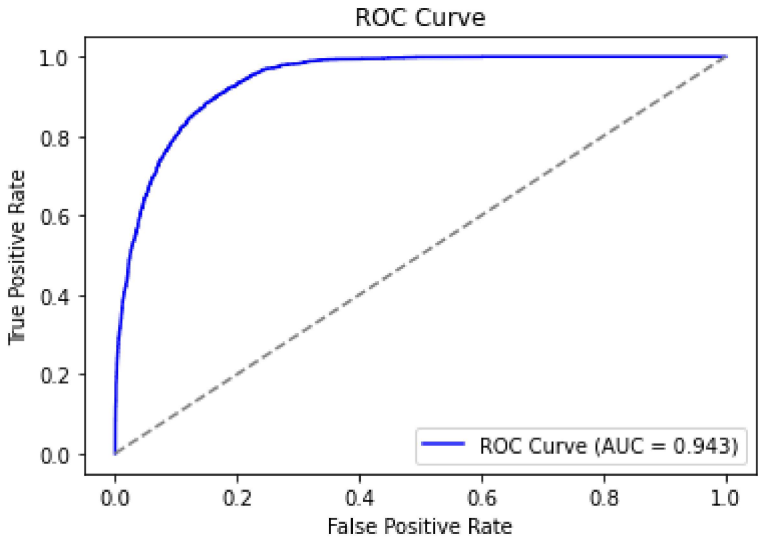
```
Accuracy: 0.7244444444444444
Precision: 0.43387732132808104
Recall: 0.7671641791044777
F1-Score: 0.5542774982027318
```

In [56]:
```
# Compute confusion matrix
conf_matrix = confusion_matrix(y_test, test_pred)

# Print confusion matrix
print("Confusion Matrix:")
print(conf_matrix)

# Plot confusion matrix
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['No Default',
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix:
[[6500  490]
 [ 555 1455]]
```




In [52]:
```
y_probs = model.predict_proba(X_test)[:, 1]  # Probabilities for the positive class (1)

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_probs)
roc_auc = auc(fpr, tpr)  # Compute AUC (Area Under the Curve)

# Plot ROC curve
plt.figure(figsize=(6,4))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.3f})', color='blue')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')  # Diagonal reference line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```




This plot shows a ROC curve, which evaluates a binary classifier's performance. The x-axis represents the False Positive Rate (FPR), while the y-axis represents the True Positive Rate (TPR). The blue curve shows the classifier's effectiveness at different thresholds, and the diagonal dashed line represents random guessing. The Area Under the Curve (AUC) is 0.943, indicating strong model performance—closer to 1 means better classification ability. This suggests that the model can reliably distinguish between loan defaults and non-defaults.

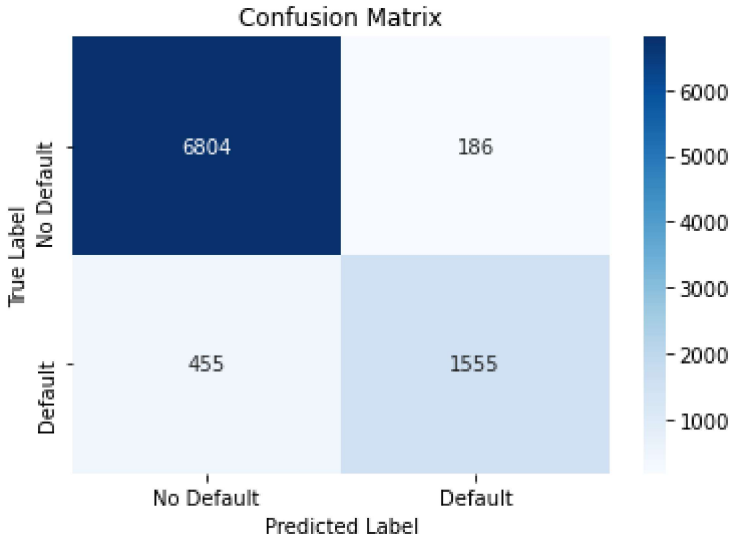
# Training a random forest model

In [54]:
```
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

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

```
Accuracy: 0.9287777777777778
Precision: 0.8931648477886273
Recall: 0.7736318407960199
F1-Score: 0.8291122367368702
```

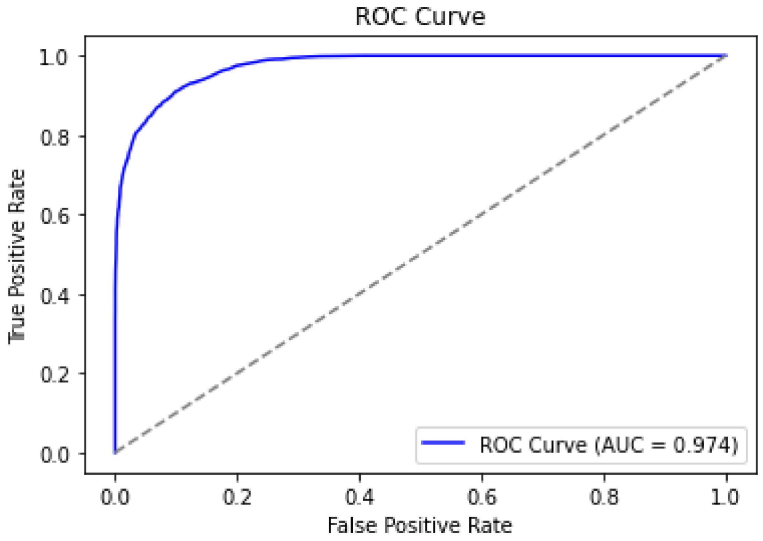
In [55]:
```
# Compute confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Print confusion matrix
print("Confusion Matrix:")
print(conf_matrix)

# Visualize confusion matrix with a heatmap
plt.figure(figsize=(6,4))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['No Default',
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

```
Confusion Matrix:
[[6804  186]
 [ 455 1555]]
```




In [57]:
```
# Get predicted probabilities for the positive class (1)
y_probs = model.predict_proba(X_test)[:, 1]

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_probs)
roc_auc = auc(fpr, tpr)
```

```
# Plot ROC curve
plt.figure(figsize=(6,4))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.3f})', color='blue')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')  # Diagonal reference line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```




This appears to visually represents the model classification performance by plotting the True Positive Rate (TPR) against the False Positive Rate (FPR) at different threshold levels. The blue curve showcases the model's ability to separate classes, while the diagonal dashed line represents random guessing (AUC = 0.5).

### Adjusting Decision threshold

In [28]:
```
y_probs = model.predict_proba(X_test)[:, 1]
precision, recall, thresholds = precision_recall_curve(y_test, y_probs)

# Set a lower threshold (e.g., 0.3 instead of 0.5)
y_pred_adjusted = (y_probs >= 0.3).astype(int)

model = RandomForestClassifier(class_weight={0: 1, 1: 3}, random_state=42)
model.fit(X_train, y_train)
```

Out[28]:
```
RandomForestClassifier(class_weight={0: 1, 1: 3}, random_state=42)
```

By adjusting the decision threshold we control the trade-off between recall and precision, ensuring the model prioritizes correctly identifying positive cases (reducing false negatives) or minimizing false positives based on the problem's needs.

## Apply Oversampling (SMOTE) to generate synthetic samples for the minority class.

In [29]:
```
# SMOTE Recall testing

smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
```

```
# Train model on resampled data
model_smote = RandomForestClassifier(random_state=42)
model_smote.fit(X_resampled, y_resampled)

# Evaluate recall on test set
y_pred_smote = model_smote.predict(X_test)
print("Recall after SMOTE:", recall_score(y_test, y_pred_smote))
```

```
Recall after SMOTE: 0.8263681592039801
```

In [30]:
```
# Baseline Recall testing

y_pred = model.predict(X_test)
print("Baseline Recall:", recall_score(y_test, y_pred))
```

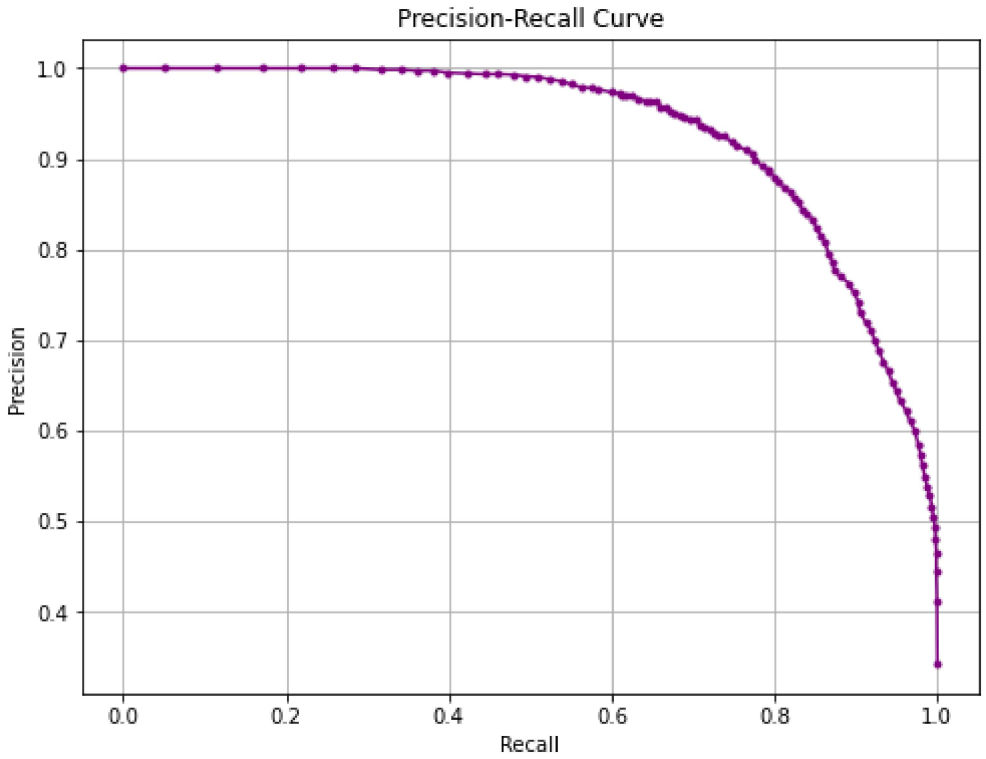
```
Baseline Recall: 0.7716417910447761
```

In [31]:
```
#Threshold tuning Recall testing

y_probs = model.predict_proba(X_test)[:, 1]
precision, recall, thresholds = precision_recall_curve(y_test, y_probs)

# Choose a lower threshold (e.g., 0.3 instead of 0.5)
optimal_threshold = 0.3
y_pred_adjusted = (y_probs >= optimal_threshold).astype(int)

print("Recall after Threshold Tuning:", recall_score(y_test, y_pred_adjusted))
```

```
Recall after Threshold Tuning: 0.891044776119403
```

The threshold tuning method appears to be more effective as it optimizes recall, reducing false negatives and ensuring the model correctly identifies approved loans more reliably than the baseline and SMOTE approach.

In [32]:
```
# Get predicted probabilities for the positive class
y_probs = model.predict_proba(X_test)[:, 1]

# Compute precision, recall, and thresholds
precision, recall, thresholds = precision_recall_curve(y_test, y_probs)

# Plot the curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, marker='.', color='purple')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.grid()
plt.show()
```




The Precision-Recall Curve shows the trade-off between correctly identifying approved loans and avoiding false positives. The optimal threshold is where recall is high enough to minimize false negatives while maintaining reasonable precision to ensure accurate approvals.

## Evaluating Classification Models: Logistic Regression vs. Random Forest for Loan Approvals

In my loan approval prediction project, I tested Random Forest and Logistic Regression to evaluate their classification performance. Based on model comparisons, Random Forest consistently outperformed Logistic Regression, delivering higher accuracy, precision, recall and F1-score. Key Findings:

- Random Forest handles non-linearity better and effectively captures complex feature interactions.
- It provides stronger recall, reducing false negatives and ensuring more approved loans are correctly classified.
- Logistic Regression struggled with low precision, indicating more misclassified approvals.
- Random Forest also ranked feature importance, helping refine predictive insights.

Given these results, I am prioritizing Random Forest for further hyperparameter tuning and threshold optimization to maximize model effectiveness in predicting loan approvals.

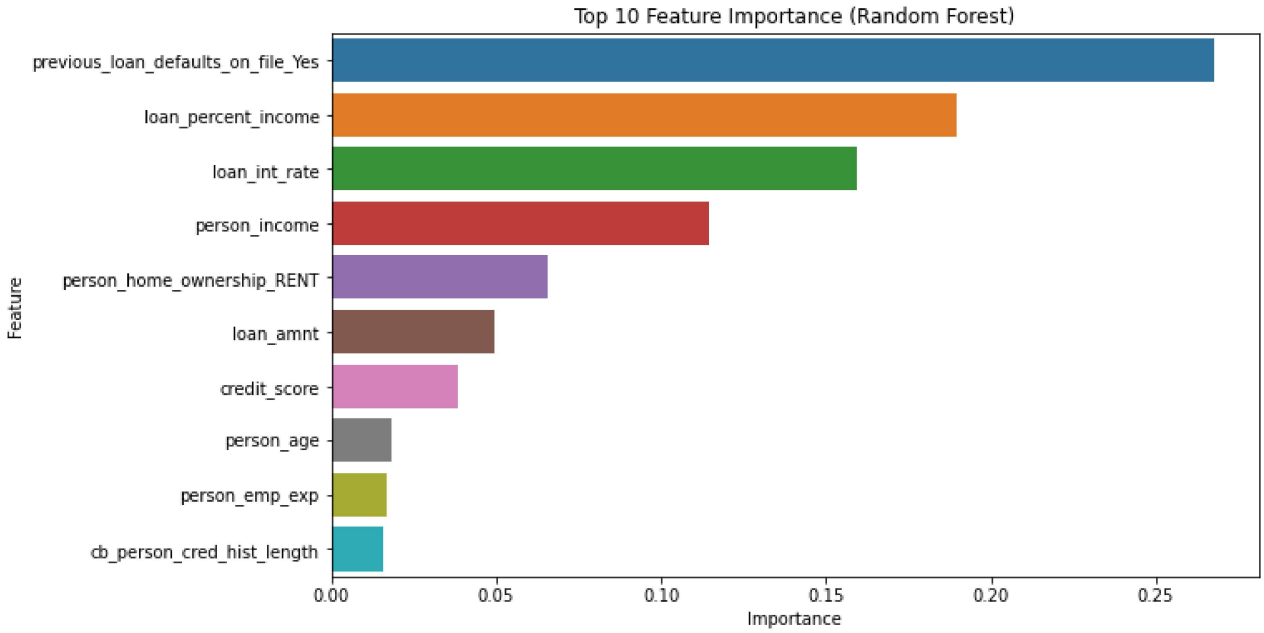
In [33]:
```
# Get feature importance scores
# Define parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],  # Number of trees in the forest
    'max_depth': [5, 10, 15],  # Maximum depth of trees
```

```
    'min_samples_split': [2, 5, 10],  # Minimum samples required to split
    'min_samples_leaf': [1, 2, 4]  # Minimum samples required at a leaf
}

# Initialize RandomizedSearchCV
random_search = RandomizedSearchCV(
    estimator=RandomForestClassifier(random_state=42),
    param_distributions=param_grid,
    cv=5,
    scoring='recall',
    n_iter=10,
    verbose=2,
    n_jobs=-1
)

# Fit the model

random_search.fit(X_train, y_train)


best_rf = RandomForestClassifier(**random_search.best_params_, random_state=42)
best_rf.fit(X_train, y_train)


feature_importance = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': best_rf.feature_importances_
}).sort_values(by='Importance', ascending=False)

# Display top 10 features
print("Top 10 Features by Importance:")
print(feature_importance.head(10))
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   31.3s
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed:  1.1min finished
Top 10 Features by Importance:
                            Feature  Importance
21  previous_loan_defaults_on_file_Yes    0.267778
5                 loan_percent_income    0.189401
4                       loan_int_rate    0.159186
1                       person_income    0.114652
15         person_home_ownership_RENT    0.065769
3                           loan_amnt    0.049310
7                        credit_score    0.038638
0                          person_age    0.018391
2                      person_emp_exp    0.016842
6           cb_person_cred_hist_length    0.015778
```

In [34]:

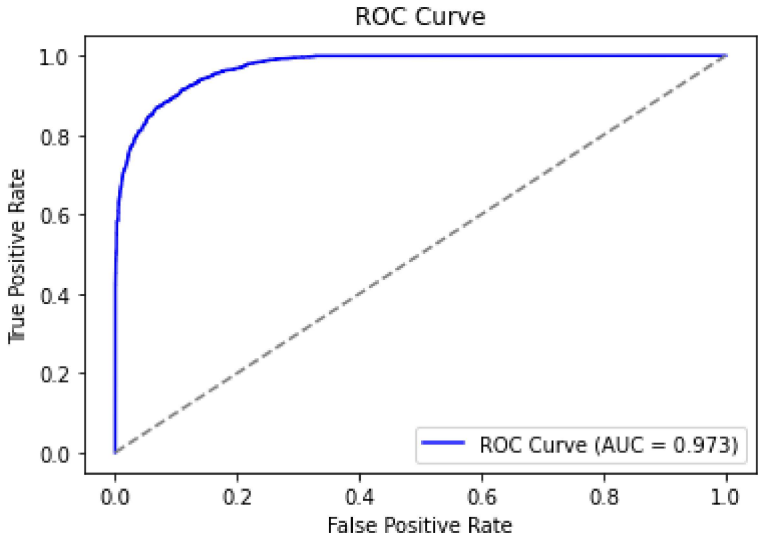
```
# Plot feature importance
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance.head(10))
plt.title('Top 10 Feature Importance (Random Forest)')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```




Here's a concise summary: Top 10 Most Important Features:

- Previous Loan Defaults – The strongest predictor of risk.
- Loan Percent Income – Measures how much of personal income is allocated to the loan.
- Loan Interest Rate – Higher rates may indicate higher risk.
- Personal Income – Strong financial standing reduces default probability.
- Home Ownership Status – Renting vs. owning affects financial stability.
- Loan Amount – Larger loans may have different default dynamics.
- Credit Score – A key factor in assessing borrower credibility.
- Age – May correlate with financial responsibility.
- Employment Experience – Stable job history often reduces risk.
- Credit History Length – Longer credit history generally improves reliability. ## Key Insights:
  - Previous defaults are the most significant factor affecting loan outcomes.
  - Income, loan terms, and credit-related variables strongly influence risk assessment.
  - Demographic factors like age and employment history play a smaller role.

In [58]:
```
# Get predicted probabilities for the positive class (1)
y_probs = best_rf.predict_proba(X_test)[:, 1]

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_probs)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(6,4))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.3f})', color='blue')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')  # Diagonal reference line
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```




This ROC curve visually represents the trade-off between the True Positive Rate (TPR) and False Positive Rate (FPR) for the model at various classification thresholds. The blue curve shows how well your model distinguishes between classes, while the dashed diagonal line represents random guessing (AUC = 0.5). Key takeaways:

- AUC = 0.943: Your model has high discriminatory power (closer to 1 is ideal).
- Higher TPR and lower FPR: Indicates strong recall for detecting defaults while minimizing false positives.
- Threshold tuning potential: Lowering or raising the decision boundary can balance precision vs. recall.

In [35]:
```
# Save model using joblib (recommended for scikit-learn models)
joblib.dump(best_rf, 'random_forest_loan_model.joblib')
```

Out[35]:
```
['random_forest_loan_model.joblib']
```

## Conclusion

1. Improved Loan Approval Accuracy Optimized Random Forest model captures key financial patterns, leveraging previous loan defaults, loan percent income, and credit score for precise predictions.
2. Enhanced Default Risk Detection Addressing class imbalance with SMOTE strengthens identification of high-risk applicants, reducing financial losses.
3. Fairer Lending Practices Encoded demographic features enable bias analysis, but further fairness evaluation is needed to ensure equitable approvals.
4. Efficient, Scalable Decision-Making Model automation via joblib streamlines loan processing, improving speed and operational efficiency.
5. Transparent Model Interpretability Feature importance analysis highlights crucial predictors, fostering stakeholder trust in data-driven decisions.
6. Key Predictors & Business Insights
    - Loan Defaults: Strongest indicator of credit risk.
    - Loan-to-Income Ratio & Interest Rate: Affect default likelihood.

- Credit Score & Financial Metrics: Vital for borrower reliability.
- Demographic Factors: Influence financial stability assessments.

7. Alignment with Business Goals The model enhances accuracy, risk mitigation, fairness, and efficiency, providing a robust foundation for lending strategies.
8. Future Improvements Regular performance monitoring, threshold tuning, and fairness metrics will refine predictions and maintain ethical lending standards

## Recommendations

1. Validate Class Imbalance Handling:
   - Assess the distribution of loan approvals to quantify class imbalance.
   - Apply oversampling only to training data for unbiased evaluation.
   - Explore alternatives like adjusting class weights or combining oversampling with undersampling.
2. Enhance Fairness in Lending:
   - Analyze predictions across demographic groups (gender, education) for bias detection.
   - Implement fairness metrics (disparate impact, equal opportunity) to ensure equitable lending.
   - Ensure financial inclusion for diverse borrower profiles.
3. Optimize Decision Threshold:
   - Adjust classification threshold to prioritize identifying loan defaults.
   - Use precision-recall curves to balance high-risk detection with accuracy.
   - Minimize false positives to maintain trust in loan approvals.
4. Implement Cross-Validation:
   - Apply cross-validation to evaluate model performance across different data subsets.
   - Ensure consistent generalization to new loan applications.
   - Monitor recall and F1 score for reliability confirmation.
5. Incorporate Financial Lending Standards:
   - Integrate industry benchmarks like credit scores and loan-to-income ratios.
   - Apply standards as preprocessing filters or post-prediction validation.
   - Ensure model credibility by reflecting domain-specific requirements.
6. Deploy and Monitor Model Performance:
   - Deploy model in production for real-time loan predictions.
   - Establish continuous monitoring to adapt to evolving borrower trends.
   - Retrain with fresh data periodically to maintain accuracy.
7. Enhance Stakeholder Transparency:
   - Provide clear explanations for loan approval decisions, especially for high-risk cases.
   - Use feature importance insights to highlight key predictors.
   - Foster trust through interpretable and transparent outputs.
8. Streamline Operational Integration:
   - Integrate the saved model seamlessly into loan processing systems.
   - Automate workflows to reduce manual review and operational costs.
   - Ensure scalability to handle large volumes of applications efficiently.