Project 4 Loan Prediction

Import necessary libraries

In [3]:
```
# Import 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
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classificatio
```

Step 1: Load the dataset

In [6]:
```
# Step 1: Load the dataset
data_path = r"C:\Users\Hello\Desktop\Loan_Predication.csv"
data = pd.read_csv(data_path)
```

Step 2: Explore the data (EDA)

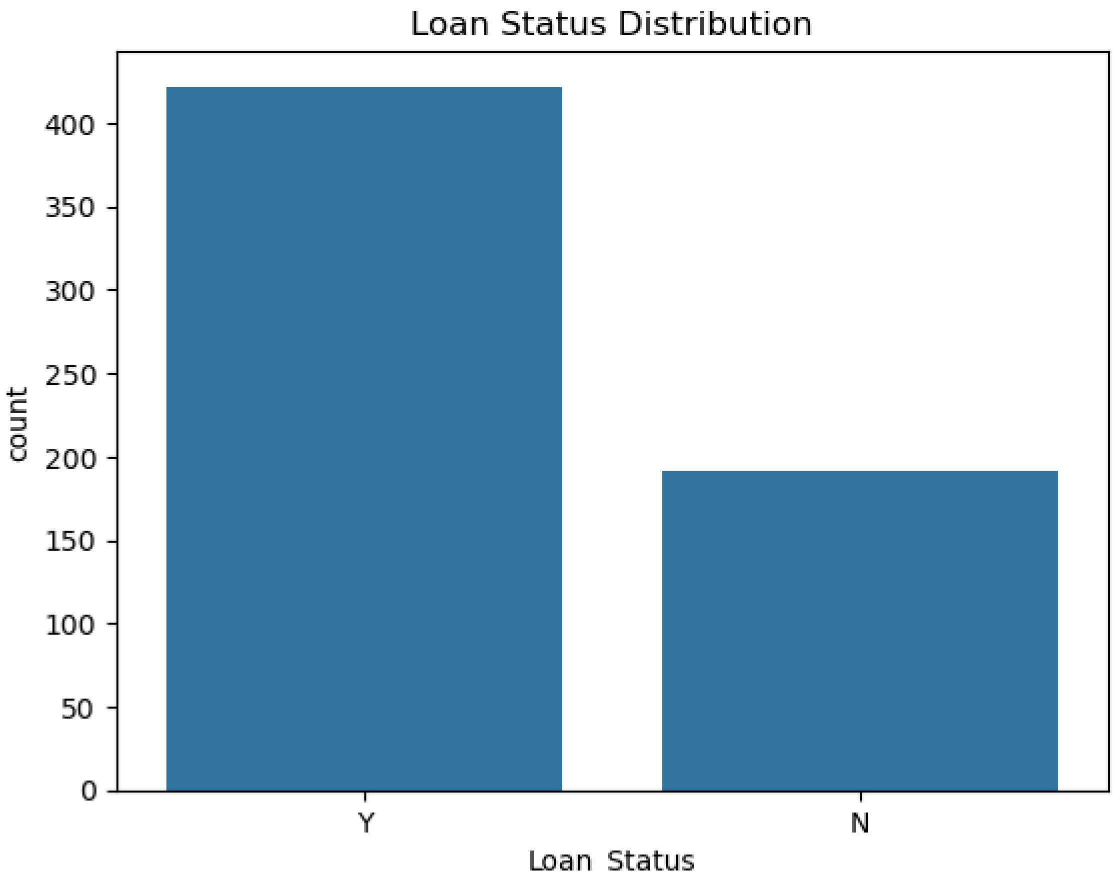
In [11]:
```
# Step 2: Explore the data (EDA)
print("Dataset Overview:")
print(data.info())
print("\nMissing Values:")
print(data.isnull().sum())
print("\nSummary Statistics:")
print(data.describe())
```

```
Dataset Overview:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            614 non-null    object 
 1   Gender             601 non-null    object 
 2   Married            611 non-null    object 
 3   Dependents         599 non-null    object 
 4   Education          614 non-null    object 
 5   Self_Employed      582 non-null    object 
 6   ApplicantIncome    614 non-null    int64  
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object 
 12  Loan_Status        614 non-null    object 
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
None

Missing Values:
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64

Summary Statistics:
       ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
count       614.000000         614.000000  592.000000         600.00000   
```

```
mean       5403.459283        1621.245798  146.412162         342.00000
std        6109.041673        2926.248369   85.587325          65.12041
min         150.000000           0.000000    9.000000          12.00000
25%        2877.500000           0.000000  100.000000         360.00000
50%        3812.500000        1188.500000  128.000000         360.00000
75%        5795.000000        2297.250000  168.000000         360.00000
max       81000.000000       41667.000000  700.000000         480.00000

       Credit_History
count      564.000000
mean         0.842199
std          0.364878
min          0.000000
25%          1.000000
50%          1.000000
75%          1.000000
max          1.000000
```

In [13]:

```
# Visualizations
sns.countplot(x='Loan_Status', data=data)
plt.title("Loan Status Distribution")
plt.show()
```



 # Step 3: Handle missing values

In [16]:
```
# Step 3: Handle missing values
imputer = SimpleImputer(strategy='most_frequent')
data_imputed = pd.DataFrame(imputer.fit_transform(data), columns=data.columns)
```

Step 4: Encode categorical variables

In [19]:
```
# Step 4: Encode categorical variables
categorical_cols = data_imputed.select_dtypes(include=['object']).columns
label_encoders = {}
for col in categorical_cols:
```

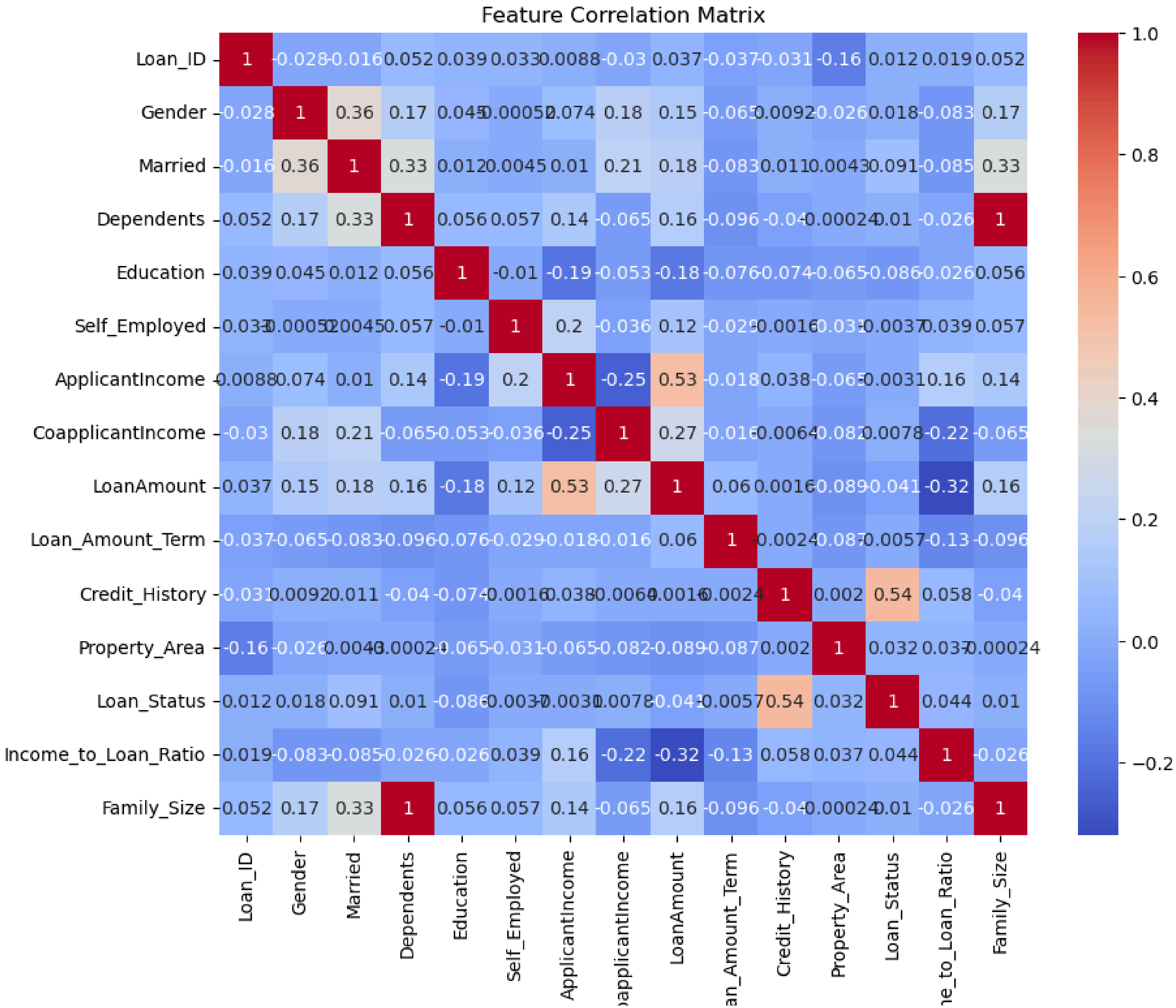
```
    label_encoders[col] = LabelEncoder()
    data_imputed[col] = label_encoders[col].fit_transform(data_imputed[col])
```

Step 5: Feature engineering

In [22]:
```
# Step 5: Feature engineering
data_imputed['Income_to_Loan_Ratio'] = data_imputed['ApplicantIncome'] / (data_imputed['LoanAmount'] + 1)
data_imputed['Family_Size'] = data_imputed['Dependents'].replace('3+', 3).astype(int) + 1
```

Step 6: Feature selection

In [25]:
```
# Step 6: Feature selection
corr_matrix = data_imputed.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title("Feature Correlation Matrix")
plt.show()
```


Feature Correlation Matrix
Loan_ID
Gender
Married
Dependents
Education
Self_Employed
ApplicantIncome
CoapplicantIncome
LoanAmount
Loan_Amount_Term
Credit_History
Property_Area
Loan_Status
Income_to_Loan_Ratio
Family_Size
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

Cc Lo Incon

Step 7: Split data

In [28]:
```
# Step 7: Split data
X = data_imputed.drop('Loan_Status', axis=1)
y = data_imputed['Loan_Status']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Step 8: Build a baseline model (Logistic Regression)

In [31]:
```
# Step 8: Build a baseline model (Logistic Regression)
logistic_model = LogisticRegression(max_iter=1000)
logistic_model.fit(X_train, y_train)
y_pred_baseline = logistic_model.predict(X_test)
```

```
C:\Users\Hello\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:469: ConvergenceWarning: lbfgs failed to
converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Step 9: Model evaluation (Logistic Regression)

In [34]:
```
# Step 9: Model evaluation (Logistic Regression)
print("Logistic Regression Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_baseline))
print("Precision:", precision_score(y_test, y_pred_baseline))
print("Recall:", recall_score(y_test, y_pred_baseline))
print("F1 Score:", f1_score(y_test, y_pred_baseline))
```

```
Logistic Regression Performance:
Accuracy: 0.7837837837837838
Precision: 0.7564102564102564
Recall: 0.9833333333333333
F1 Score: 0.855072463768116
```

Step 10: Model tuning (Random Forest)

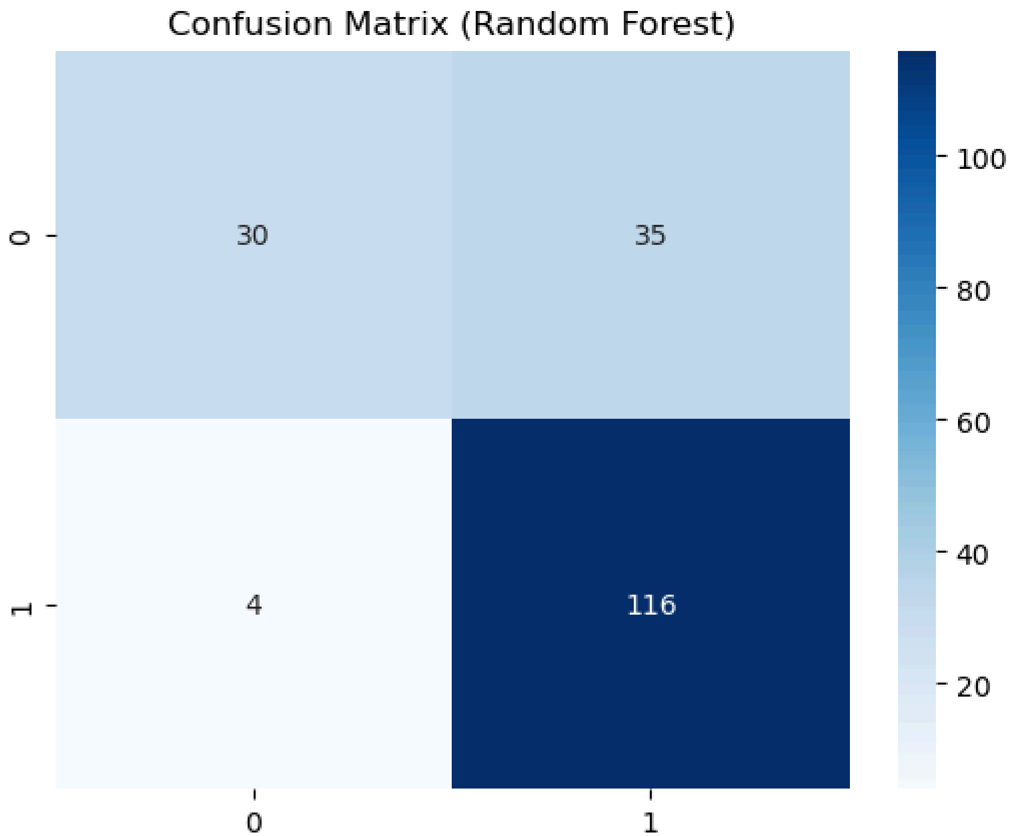
In [37]:
```
# Step 10: Model tuning (Random Forest)
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)
```

In [39]:
```
# Model evaluation (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))
```

```
Random Forest Performance:
Accuracy: 0.7891891891891892
Precision: 0.7682119205298014
Recall: 0.9666666666666667
F1 Score: 0.8560885608856088
```

In [41]:
```
# Compare results
print("\nClassification Report (Random Forest):")
print(classification_report(y_test, y_pred_rf))
print("Confusion Matrix:")
sns.heatmap(confusion_matrix(y_test, y_pred_rf), annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix (Random Forest)")
plt.show()
```

```
Classification Report (Random Forest):
              precision    recall  f1-score   support

           0       0.88      0.46      0.61        65
           1       0.77      0.97      0.86       120

    accuracy                           0.79       185
   macro avg       0.83      0.71      0.73       185
weighted avg       0.81      0.79      0.77       185

Confusion Matrix:
```



Step 11: Analyze results

Step 11: Analyze results

print("Logistic Regression vs Random Forest:") print(f"Logistic Regression F1: {f1_score(y_test, y_pred_baseline)}") print(f"Random Forest F1: {f1_score(y_test, y_pred_rf)}")

In [ ]: