In [155…

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, r
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, c
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
```

In [156…

```
Test_data = pd.read_csv(r"C:\Users\HP\Downloads\Assignment - Bounce.xlsx - Data.csv")
```

In [157…

```
Test_data['default_flag'] = data['collection_due_bucket'].apply(lambda x: 1 if x != 'N
```

In [158…

```
Test_data.Region
```

Out[158]:

```
0             Bali
1             Bali
2             Bali
3             Bali
4             Bali
           ...
19905          USA
19906          USA
19907    Sri Lanka
19908         Bali
19909    Sri Lanka
Name: Region, Length: 19910, dtype: object
```

In [159…

```
Test_data['disbursed_date'] = pd.to_datetime(Test_data['disbursed_date'])
Test_data['emi_due_date'] = pd.to_datetime(Test_data['emi_due_date'])
Test_data['emi_paid_date'] = pd.to_datetime(Test_data['emi_paid_date'])

# Create new columns
# Calculate EMI amount
Test_data['emi_amount'] = Test_data['principal'] / Test_data['productTenure']
# Create default flag
Test_data['default_flag'] = Test_data['collection_due_bucket'].apply(lambda x: 1 if x

# Handle missing values
Test_data = Test_data.dropna()
```

Data Preprocessing

Dates converted to datetime format for easier calculations.

New Columns:

emi_amount: Calculated as principal / productTenure.\ default_flag: Created based on collection_due_bucket (1 if overdue, otherwise 0).\ days_delayed: Difference between emi_paid_date and emi_due_date.\

In [160…

```
print("Descriptive Statistics:")
print(Test_data.describe())
```

```
Descriptive Statistics:
            loan_id     PartnerID  SalesManagerID     principal   loan_amount  \
count  1.598100e+04  15981.000000    15981.000000  1.598100e+04  1.598100e+04
mean   2.581839e+07     69.257368      571.610663  1.625795e+06  6.067555e+07
std    9.460674e+04     60.624847      609.176784  1.103300e+06  3.748835e+07
min    2.569364e+07      9.000000       90.000000  2.345000e+05  1.200000e+07
25%    2.574360e+07     36.000000      180.000000  9.568000e+05  3.400000e+07
50%    2.580360e+07     54.000000      270.000000  1.369700e+06  5.000000e+07
75%    2.588364e+07     81.000000      900.000000  1.972300e+06  7.000000e+07
max    2.608365e+07    405.000000     3240.000000  1.255200e+07  3.000000e+08

       productTenure           IRR  installment_due  installment_number  \
count   15981.000000  15981.000000     1.598100e+04        15981.000000
mean       54.716100     30.442119     2.767761e+06            3.996183
std         6.767602      3.506106     1.607089e+06            2.766019
min        36.000000      7.500000     4.645000e+05            1.000000
25%        48.000000     27.000000     1.761300e+06            2.000000
50%        60.000000     30.500000     2.498100e+06            3.000000
75%        60.000000     33.000000     3.310700e+06            6.000000
max        60.000000     40.000000     1.681600e+07           16.000000

        cibil_score  ...         enq_3m        enq_6m   utilization  \
count  15981.000000  ...  15981.000000  15981.000000  15981.000000
mean     743.775233  ...      9.483449     16.147237      0.706045
std       33.684923  ...      6.018988      9.819454      0.199564
min       -1.000000  ...      0.000000      0.000000     -0.011186
25%      729.000000  ...      5.000000      9.000000      0.627855
50%      745.000000  ...      8.000000     14.000000      0.748473
75%      761.000000  ...     13.000000     22.000000      0.835208
max      830.000000  ...     55.000000     85.000000      1.587674

            abb_90d  principaloutstandingamt  principalPaid    emipaidamt  \
count  1.598100e+04             1.598100e+04   1.598100e+04  1.598100e+04
mean   5.356486e+07             5.445799e+07   1.228382e+07  2.772421e+06
std    2.501074e+08             3.470524e+07   1.213956e+07  1.606890e+06
min    9.208000e+05             0.000000e+00   3.665000e+05  4.746000e+05
25%    6.024300e+06             2.947610e+07   5.121500e+06  1.776400e+06
50%    1.289200e+07             4.668950e+07   8.911800e+06  2.509700e+06
75%    3.295390e+07             6.594200e+07   1.512520e+07  3.325900e+06
max    7.436371e+09             3.000000e+08   2.000000e+08  1.681600e+07

       avrg_amt_credit_3m_monthly  default_flag     emi_amount
count                1.598100e+04  15981.000000   15981.000000
mean                 3.581655e+08      0.137789   31114.672687
std                  8.794970e+08      0.344689   24659.113756
min                  1.000000e+02      0.000000    3908.333333
25%                  7.009238e+07      0.000000   16478.333333
50%                  1.339196e+08      0.000000   24085.416667
75%                  3.049175e+08      0.000000   36148.333333
max                  1.968550e+10      1.000000  348666.666667

[8 rows x 21 columns]
```

Descriptive Statistics

Loan Amounts:

Mean: 60,675,550. Range: From 12,000,000 to 300,000,000.

CIBIL Scores:

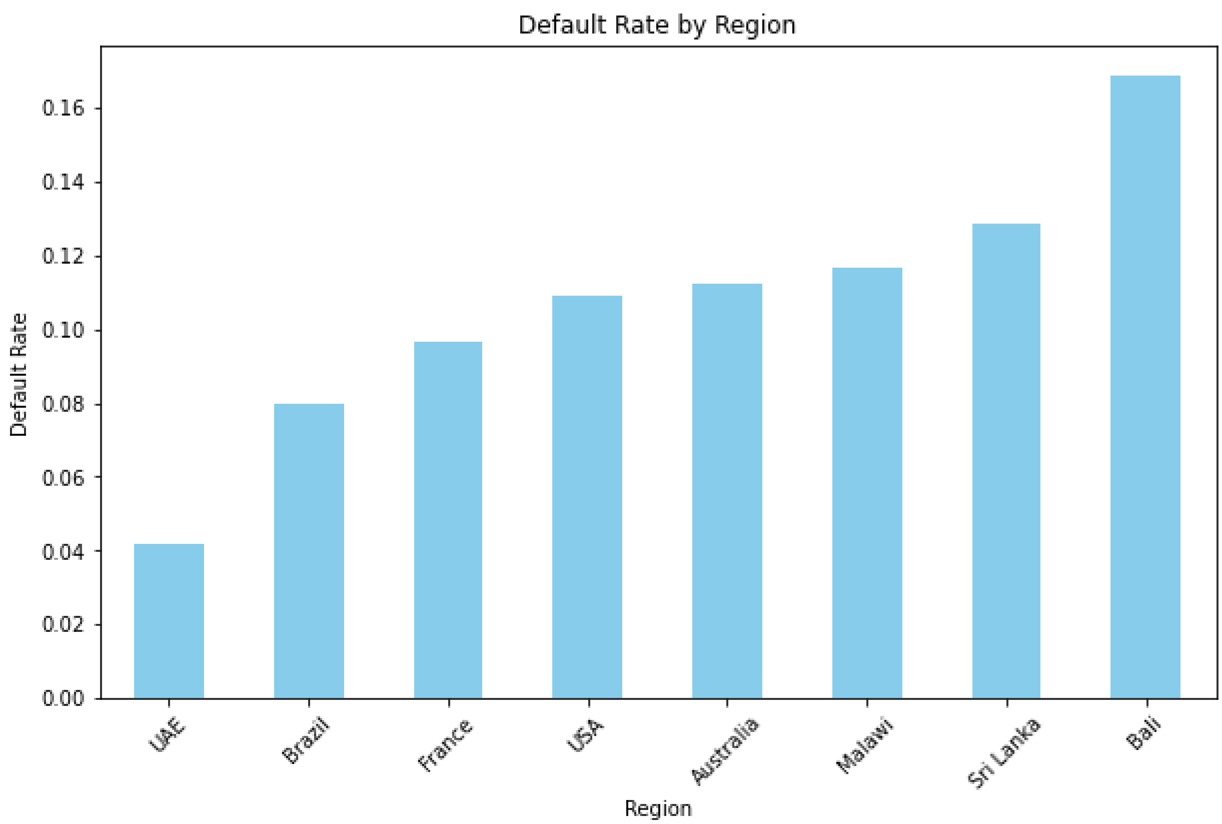
Mean: 743.77. Min: -1 (likely a missing value). Max: 830.

Default Rate:

Mean default rate across the dataset: 13.78%.

In [161...

```
# Plot default rate by region
plt.figure(figsize=(10, 6))
default_rate_by_region = Test_data.groupby('Region')['default_flag'].mean().sort_value
default_rate_by_region.plot(kind='bar', color='skyblue')
plt.title('Default Rate by Region')
plt.ylabel('Default Rate')
plt.xlabel('Region')
plt.xticks(rotation=45)
plt.show()
```



Default rates vary significantly by region, with Bali showing the highest rate and UAE the lowest.

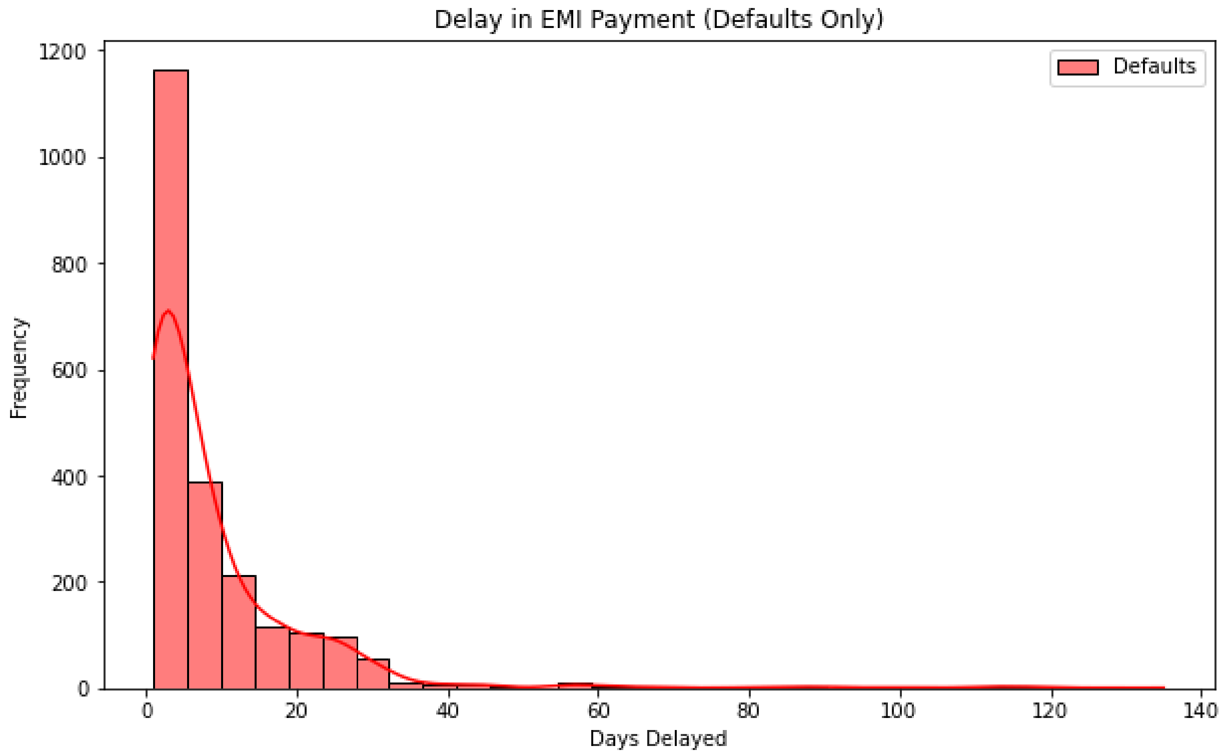
In [162...

```
import seaborn as sns
```

In [163...

```
# Analyze repayment behavior
plt.figure(figsize=(10, 6))
# Calculate days delayed, replacing NaT with 0 for valid subtraction
Test_data['days_delayed'] = (Test_data['emi_paid_date'] - Test_data['emi_due_date']).c
```

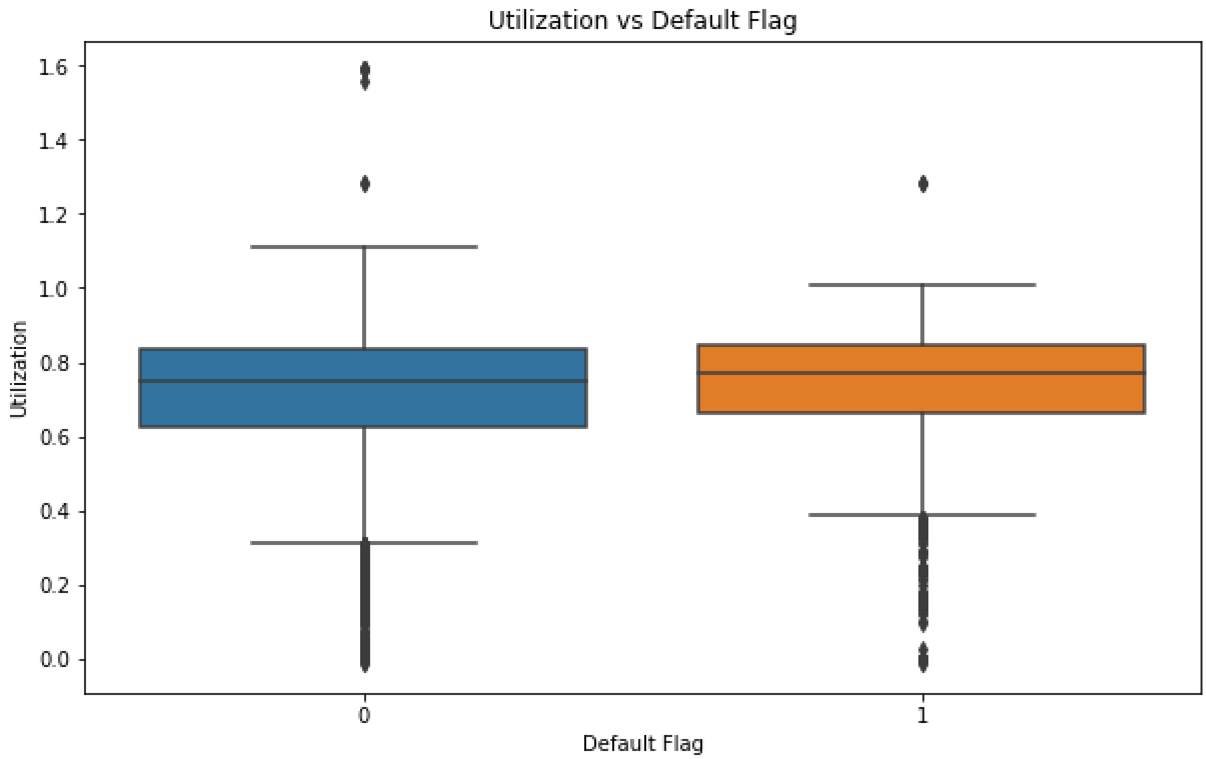
```
sns.histplot(Test_data[Test_data['default_flag'] == 1]['days_delayed'],
             bins=30, kde=True, color='red', label='Defaults')
plt.title('Delay in EMI Payment (Defaults Only)')
plt.xlabel('Days Delayed')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```



Most EMI payment delays for defaults are under 20 days, with a sharp decline beyond that.

In [164…

```
# Utilization Analysis
plt.figure(figsize=(10, 6))
sns.boxplot(x='default_flag', y='utilization', data=Test_data)
plt.title('Utilization vs Default Flag')
plt.xlabel('Default Flag')
plt.ylabel('Utilization')
plt.show()
```



Higher utilization rates may indicate a greater likelihood of default, as seen in the slightly elevated median for defaulters. However, the difference isn't dramatic, suggesting that other variables might also be critical in predicting default behavior

In [165…

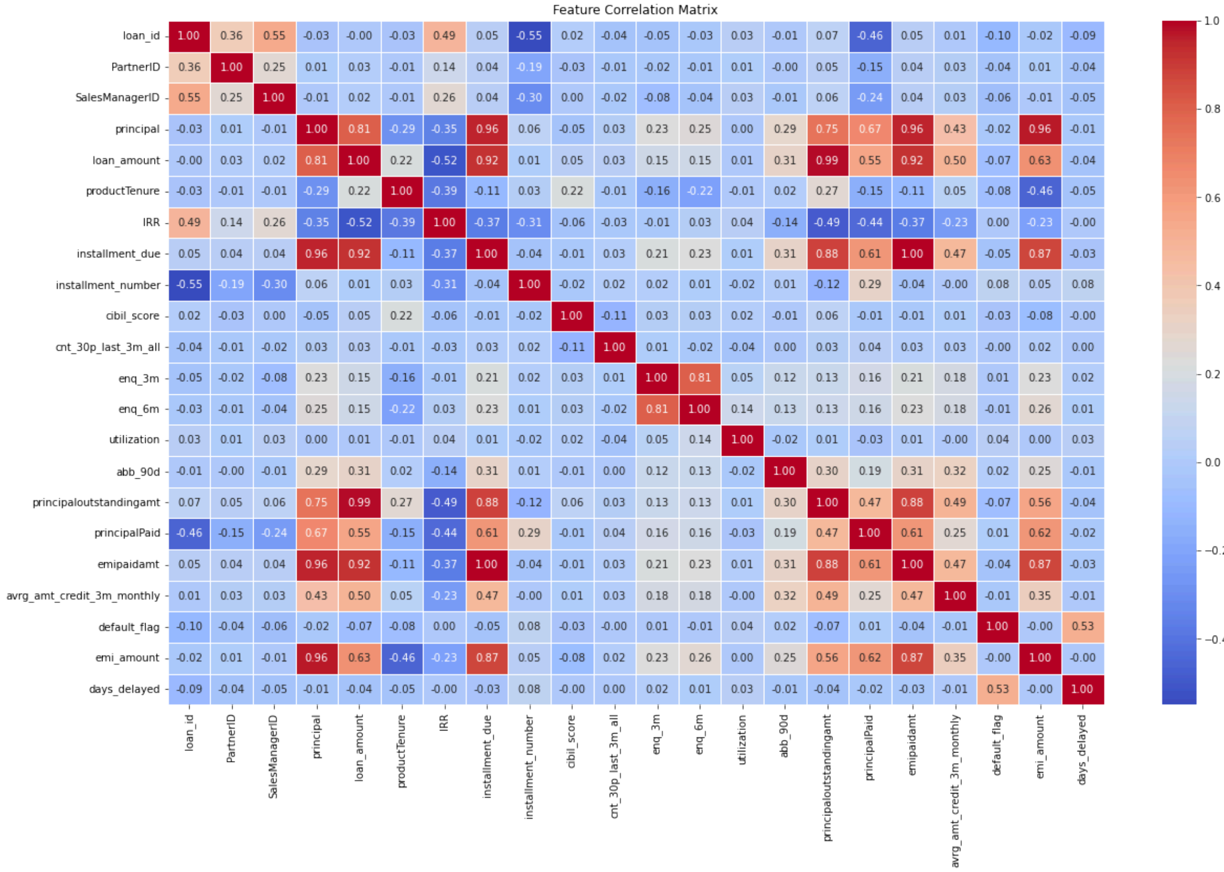
```
Test_data.columns
```

Out[165]:

```
Index(['Region', 'loan_id', 'channel', 'PartnerID', 'SalesManagerID',
       'Category', 'principal', 'loan_amount', 'productTenure', 'IRR',
       'collection_due_bucket', 'disbursed_date', 'installment_due',
       'installment_number', 'emi_due_date', 'emi_paid_date', 'cibil_score',
       'cnt_30p_last_3m_all', 'enq_3m', 'enq_6m', 'utilization', 'abb_90d',
       'principaloutstandingamt', 'principalPaid', 'emipaidamt',
       'avrg_amt_credit_3m_monthly', 'default_flag', 'emi_amount',
       'days_delayed'],
      dtype='object')
```

In [166…

```
# Feature Correlation Analysis
plt.figure(figsize=(20, 12))
corr_matrix = Test_data.corr()
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', linewidths=0.5)
plt.title('Feature Correlation Matrix')
plt.show()
```



In [167…

```
pd.DataFrame(corr_matrix).to_csv('corr_matrix.csv', index=False)
```

principal
loan_amount
productTenure
installment_due
installment_number
cibil_score
utilization
abb_90d
principaloutstandingamt
emipaidamt
default_flag
days_delayed

help in feature selection for model insatlment due is highly correlated with default flag (-0.015 to 0.015)\ principal loan_amount emipaidamt is highly correralted with installment due\ days_delayed should not be included it mean already in default

In [182…

```
categorical_columns = Test_data.select_dtypes(include=['object']).columns
Test_data = pd.get_dummies(Test_data, columns=categorical_columns, drop_first=True)
```

In [183…

```
# Prepare data for predictive modeling
features = ['installment_due', 'productTenure', 'utilization', 'abb_90d', 'principalou
target = 'default_flag'
X = Test_data[features]
y = Test_data[target]
```

In [184…

```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=

print(f"Training set size: {X_train.shape}")
print(f"Testing set size: {X_test.shape}")
```

```
Training set size: (11186, 6)
Testing set size: (4795, 6)
```

In [185…

```
# Initialize the scaler
scaler = StandardScaler()

# Fit and transform the training set, transform the test set
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [198…

```
# Initialize the Logistic Regression model
model = LogisticRegression()

# Train the model
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)
```

In [199…

```
# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the metrics
print("Model Performance:")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")

# Confusion Matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Classification Report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Model Performance:
Accuracy: 0.85
Precision: 0.75
Recall: 0.00
F1 Score: 0.01
Confusion Matrix:
[[4091    1]
 [ 700    3]]
Classification Report:
              precision    recall  f1-score   support

           0       0.85      1.00      0.92      4092
           1       0.75      0.00      0.01       703

    accuracy                           0.85      4795
   macro avg       0.80      0.50      0.46      4795
weighted avg       0.84      0.85      0.79      4795
```

The model achieves high accuracy (85%) but struggles to identify defaulters, as indicated by a low recall (0%) and F1 score (0.01), suggesting poor performance in detecting the minority class. This indicates the need for addressing class imbalance, \such as using oversampling, undersampling, or adjusting class weights.

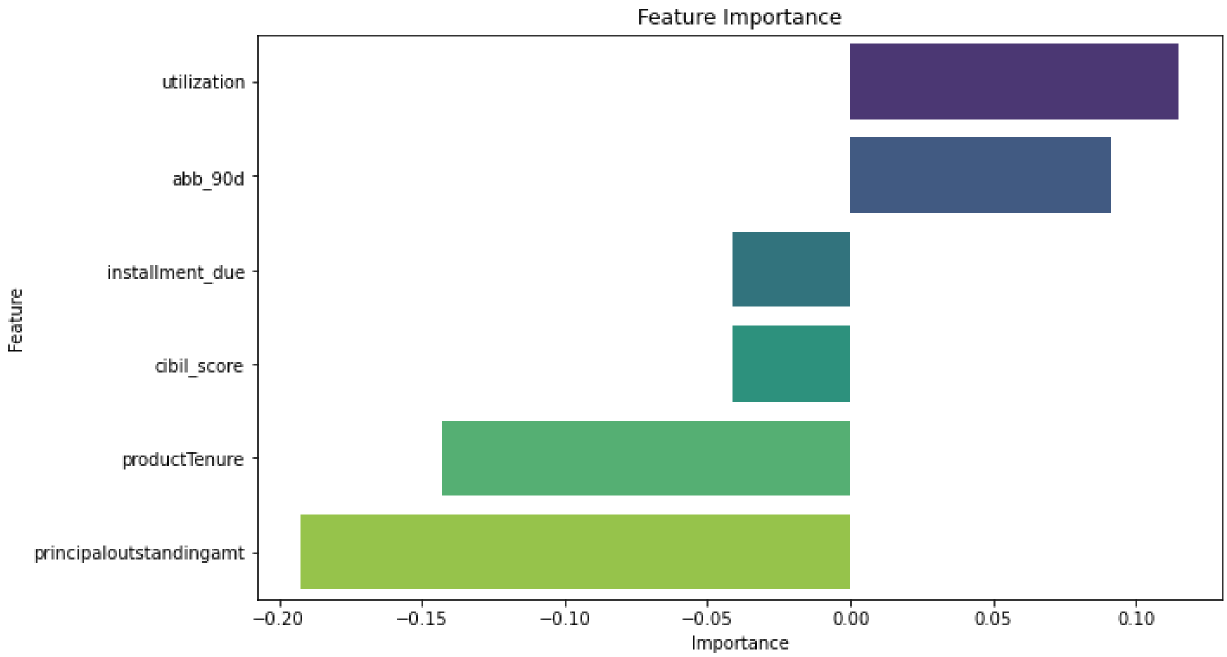
In [200…

```
feature_importance = pd.DataFrame({
    'Feature': features,
    'Coefficient': model.coef_[0]
}).sort_values(by='Coefficient', ascending=False)

print("Feature Importance:")
print(feature_importance)
```

```
Feature Importance:
                   Feature  Coefficient
2              utilization     0.114623
3                  abb_90d     0.091057
0          installment_due    -0.041186
5              cibil_score    -0.041398
1            productTenure    -0.142656
4  principaloutstandingamt    -0.192269
```

In [204…

```
# Feature Importance
importances = model.coef_[0]
feature_importance_df = pd.DataFrame({'Feature': features, 'Importance': importances})

plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df, palette='viridis'
plt.title('Feature Importance')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```



In [189…

```
from sklearn.ensemble import RandomForestClassifier

# Initialize and train Random Forest
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)

# Predict and evaluate
y_rf_pred = rf_model.predict(X_test)
print("Random Forest Accuracy:", accuracy_score(y_test, y_rf_pred))
```

```
Random Forest Accuracy: 0.8696558915537018
```

In [196…

```
# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Print the metrics
print("Model Performance:")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")

# Confusion Matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Classification Report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

```
Model Performance:
Accuracy: 0.87
Precision: 0.59
Recall: 0.38
F1 Score: 0.46
Confusion Matrix:
[[3906  186]
 [ 439  264]]
Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.95      0.93      4092
           1       0.59      0.38      0.46       703

    accuracy                           0.87      4795
   macro avg       0.74      0.67      0.69      4795
weighted avg       0.85      0.87      0.86      4795
```

The model achieved an accuracy of 87%, with a precision of 59% and a recall of 38% for the positive class, indicating moderate identification of defaults but room for improvement in recall. The F1 score of 46% highlights the trade-off between precision and recall.

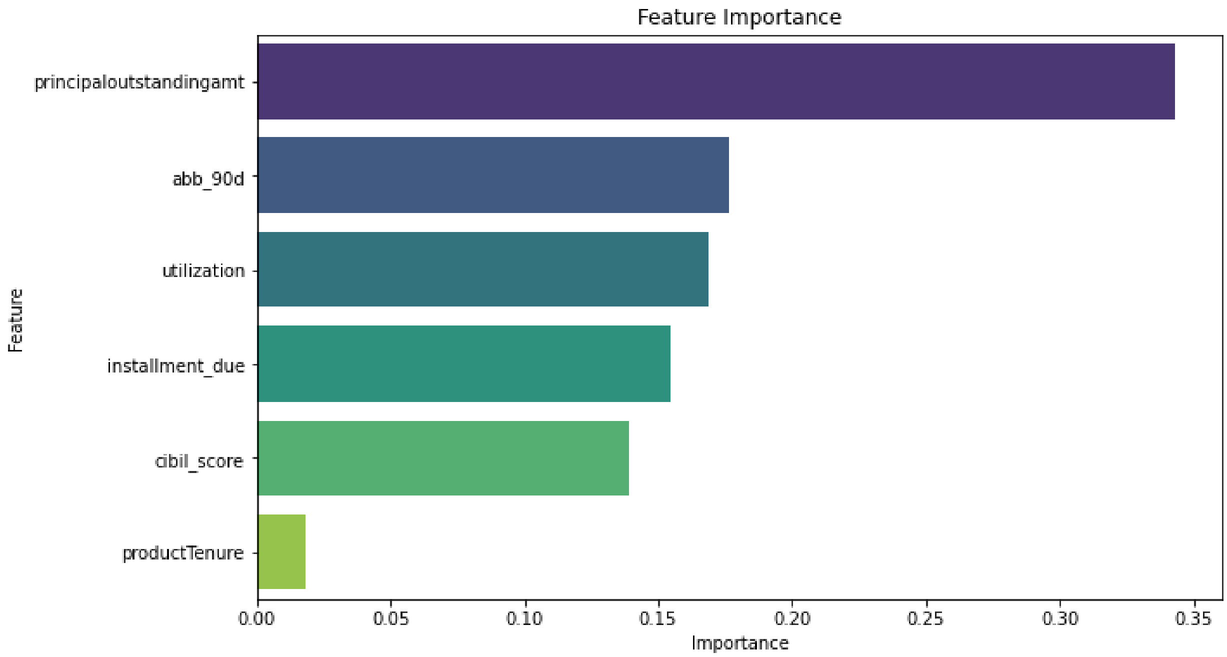
In [203…

```python
feature_importance = pd.DataFrame({
    'Feature': features,
    'Coefficient': rf_model.feature_importances_
}).sort_values(by='Coefficient', ascending=False)

print("Feature Importance:")
print(feature_importance)
```

```
Feature Importance:
                Feature  Coefficient
4  principaloutstandingamt     0.343457
3                abb_90d     0.176424
2            utilization     0.168520
0        installment_due     0.154284
5            cibil_score     0.139043
1          productTenure     0.018272
```

In [194…

```python
# Feature Importance
importances = rf_model.feature_importances_
feature_importance_df = pd.DataFrame({'Feature': features, 'Importance': importances})

plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df, palette='viridis'
plt.title('Feature Importance')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```



Random Forests focus purely on the predictive power of a feature in reducing error or impurity. Even if a feature's effect on the target is negative, it is still measured by its ability to improve the splits, which cannot result in a negative score.