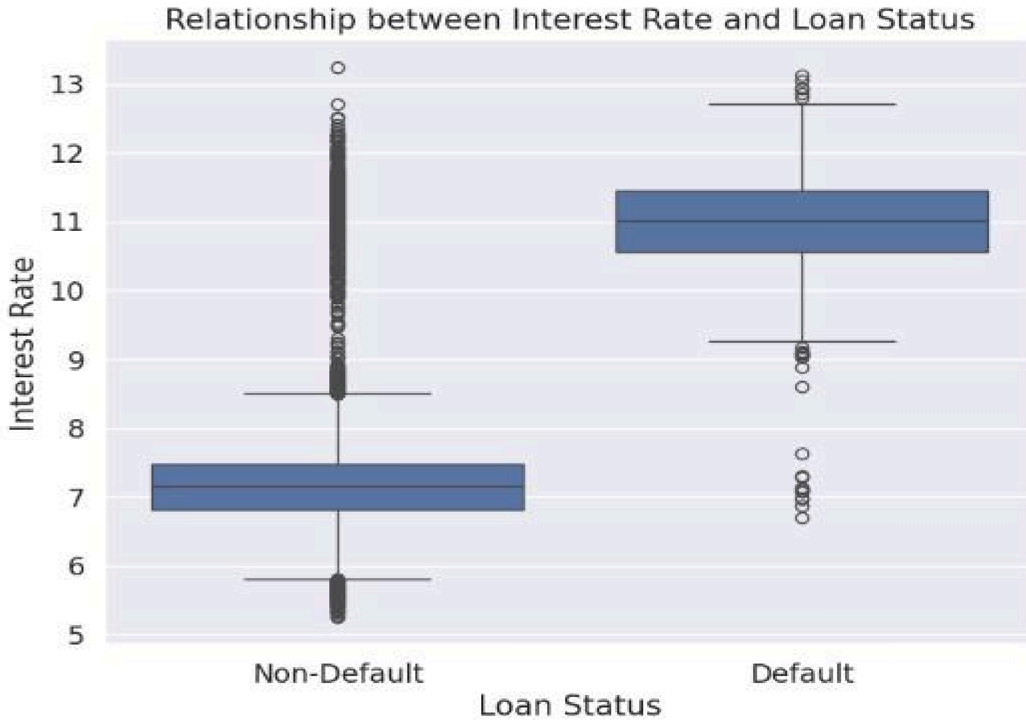
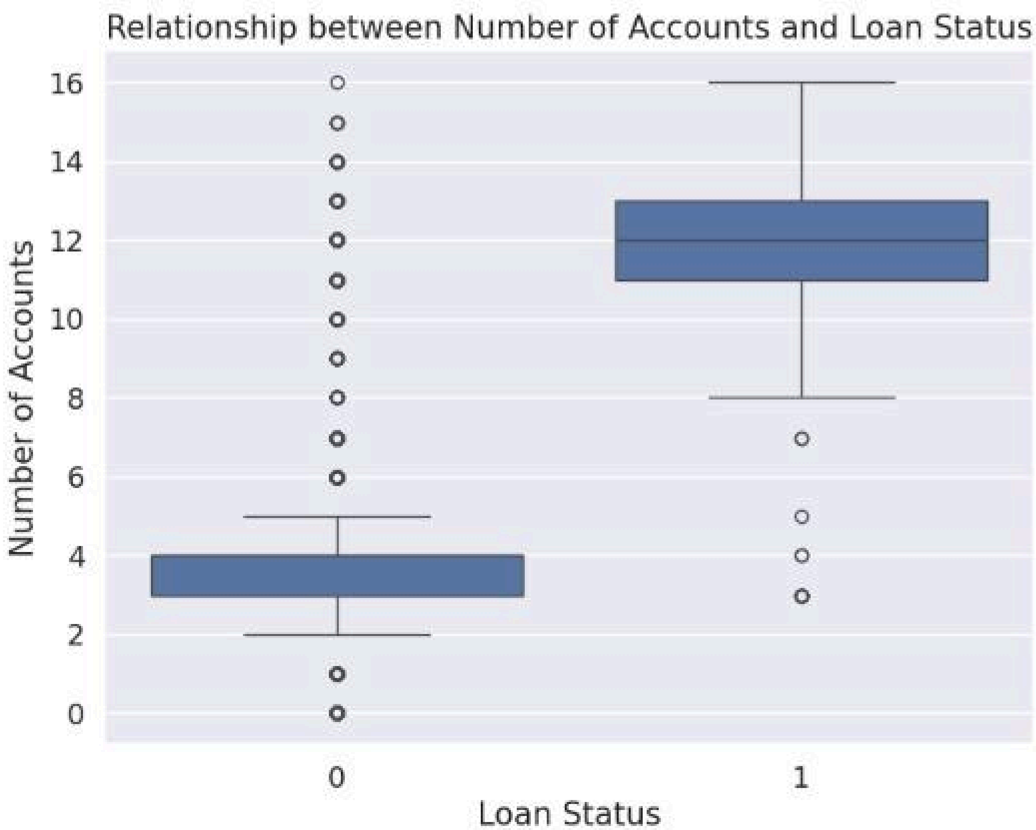
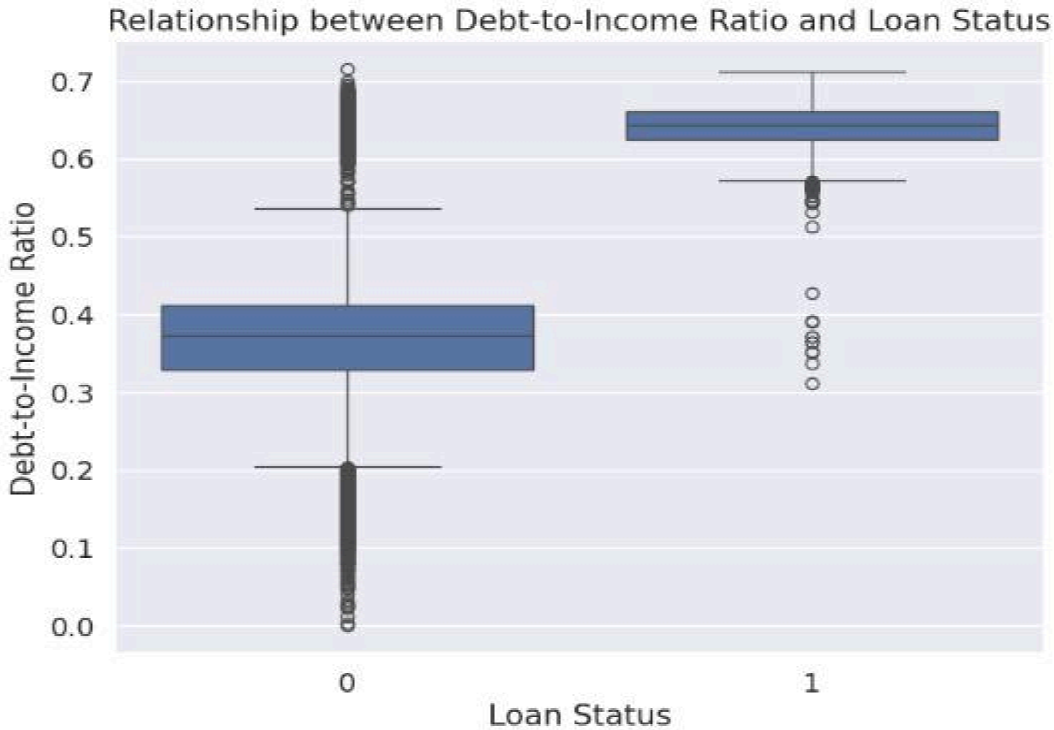
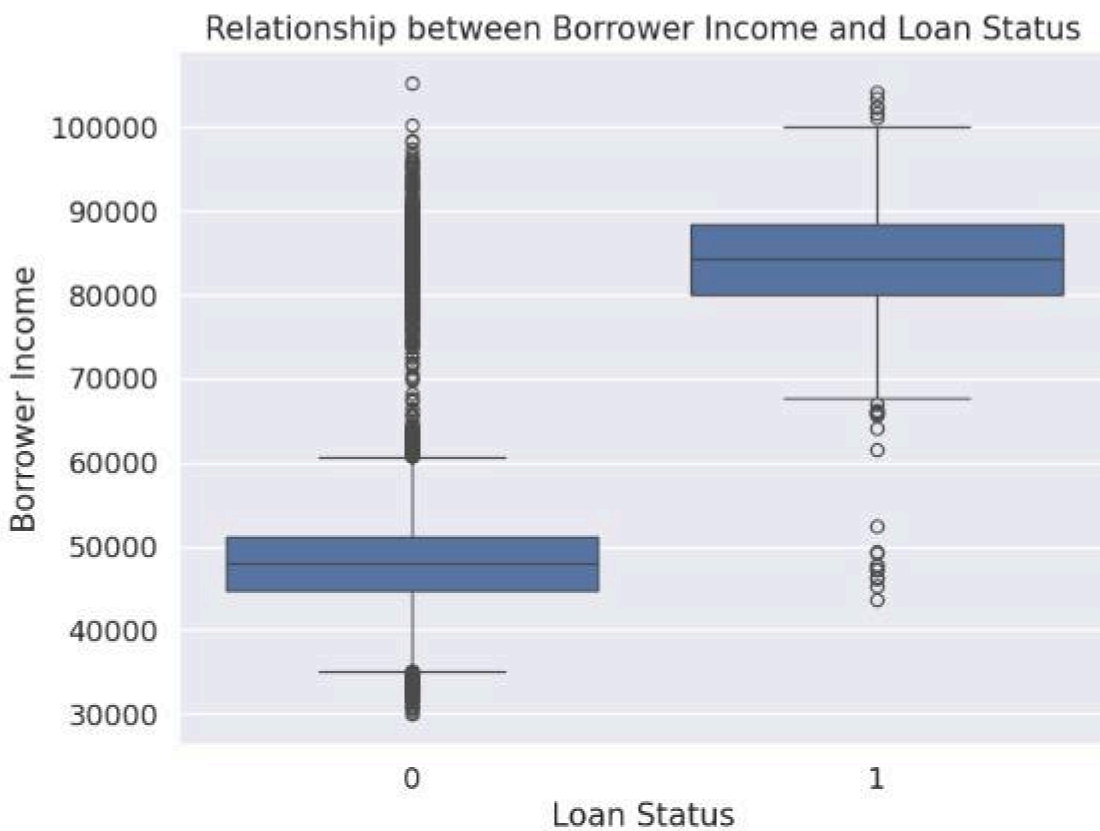
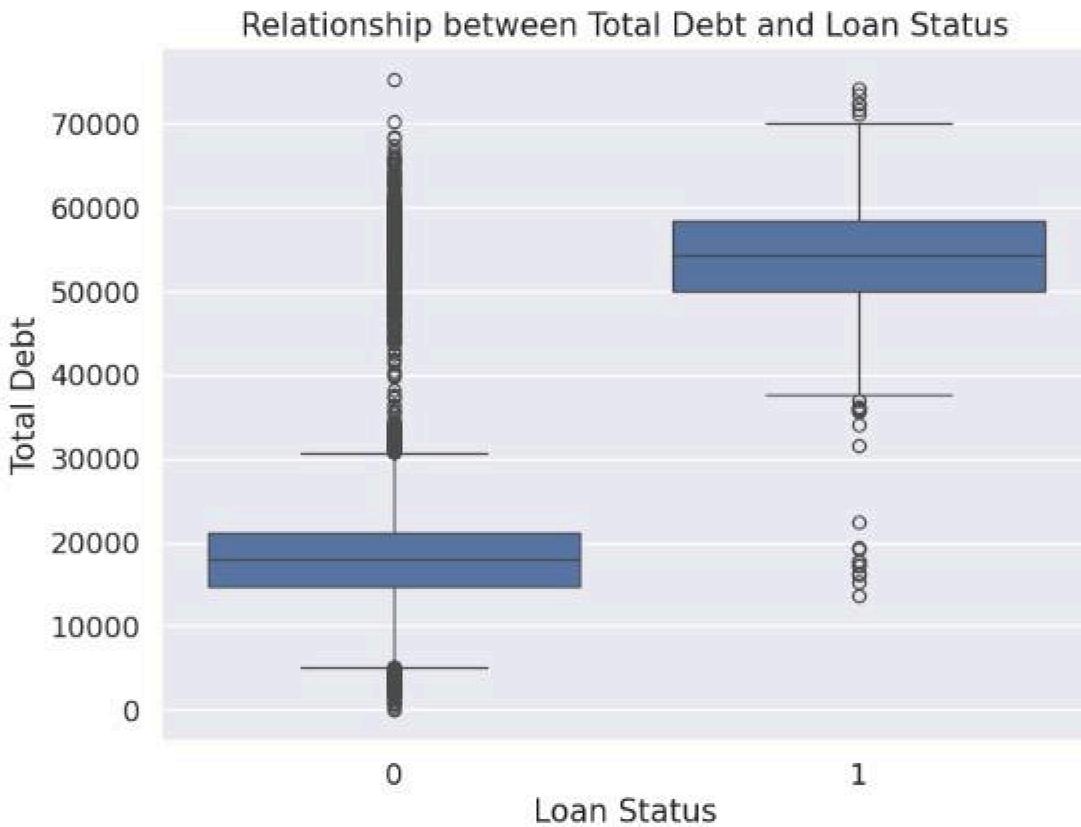
Preprocessing:

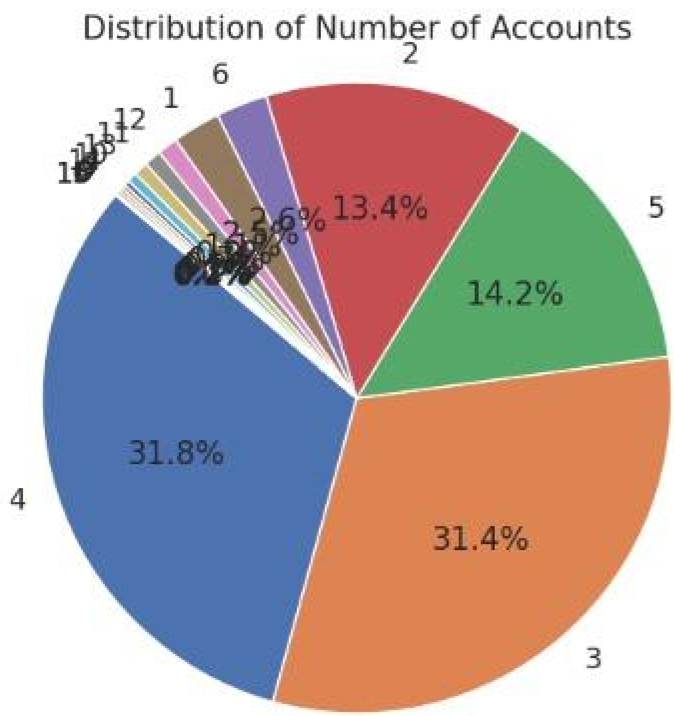





Relationship between Debt-to-Income Ratio and Loan Status
Debt-to-Income Ratio
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
1
Loan Status


Relationship between Borrower Income and Loan Status
Borrower Income
30000
40000
50000
60000
70000
80000
90000
100000
0
1
Loan Status


Relationship between Total Debt and Loan Status
70000
60000
50000
40000
30000
20000
10000
0
Total Debt
0
1
Loan Status


Distribution of Number of Accounts
2
6
1
12
5
13.4%
14.2%
31.8%
4
31.4%
3

Logistic Regression:

LOGISTIC REGRESSION

```
[17] # Initialize the Logistic Regression
     log_reg = LogisticRegression(max_iter=1000)  # You can adjust hyperparameters as needed
```

```
[18] # Train the model
     log_reg.fit(X_train, y_train)
```

```
[19] # Predictions on the test set
     y_pred = log_reg.predict(X_test)
```

```
[20] print(f"Training Score: {log_reg.score(X_train, y_train)}")
     print(f"Testing Score: {log_reg.score(X_test, y_test)}")
```

```
# Feature importance
print("Feature Coefficients:")
coefficients = log_reg.coef_[0]
for feature, coef in zip(X.columns, coefficients):
    print(f"{feature}: {coef}")
```

```
Feature Coefficients:
loan_size: 0.0048612519203814165
interest_rate: -1.807635692960808e-07
borrower_income: -0.0011996682874851838
debt_to_income: 1.7312957665138193e-05
num_of_accounts: -1.1487689698999806e-05
derogatory_marks: 0.00010070978405778181
total_debt: 0.00023806551654861275
```

```
[24] # Model evaluation
     accuracy_log_reg= accuracy_score(y_test, y_pred)
     print("Accuracy:", accuracy_log_reg)
```

```
Accuracy: 0.9924164259182832
```


```
# Model evaluation
accuracy_log_reg= accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy_log_reg)
```

```
Accuracy: 0.9924164259182832
```


```
[48] # Confusion matrix
     conf_matrix = confusion_matrix(y_test, y_pred)
     print("Confusion Matrix:")
     print(conf_matrix)
```

```
Confusion Matrix:
[[18679    80]
 [   67   558]]
```

```
[26] print("Classification Report:")
     print(classification_report(y_test, y_pred))
```

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

           0       1.00      1.00      1.00     18759
           1       0.87      0.89      0.88       625

    accuracy                           0.99     19384
   macro avg       0.94      0.94      0.94     19384
weighted avg       0.99      0.99      0.99     19384
```

```
[27] from sklearn.model_selection import cross_val_score
     scores = cross_val_score(log_reg, X, y, cv=5, scoring='accuracy')
     print("Cross-Validation Scores:", scores)
```

```
Cross-Validation Scores: [0.99123033 0.99264848 0.9920681  0.99284194 0.99142323]
```

KNN:

KNN

```
[51] # Initialize the KNN
     knn =KNeighborsClassifier(n_neighbors=100)
```

```
[52] # Train the model on the training data
     knn.fit(X_train, y_train)
```

```
      KNeighborsClassifier
KNeighborsClassifier(n_neighbors=100)
```

```
[53] y_true = y_test
     y_pred = knn.predict(X_test)
```

```
[54] print(f"Training Score: {knn.score(X_train, y_train)}")
     print(f"Testing Score: {knn.score(X_test, y_test)}")
```

```
Training Score: 0.9941532535424406
Testing Score: 0.9950990507635163
```

```
# Evaluate the model's performance
accuracy_knn= accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy_knn)
```

```
Accuracy: 0.9950990507635163
```

```
[57] # Print classification report for detailed evaluation
     print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     18759
           1       0.87      0.99      0.93       625

    accuracy                           1.00     19384
   macro avg       0.94      0.99      0.96     19384
weighted avg       1.00      1.00      1.00     19384
```

[58]

```python
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'n_neighbors': [3, 5, 7, 9],  # Number of neighbors to consider
    'metric': ['euclidean', 'manhattan']  # Distance metric
}

# Initialize GridSearchCV
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')

# Perform grid search to find the best hyperparameters
grid_search.fit(X_train_scaled, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# Train the classifier with the best hyperparameters
best_knn = KNeighborsClassifier(**best_params)
best_knn.fit(X_train_scaled, y_train)

# Make predictions on the test data
y_pred = best_knn.predict(X_test_scaled)

# Evaluate the model's performance
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Print classification report for detailed evaluation
print(classification_report(y_test, y_pred))
```

```
Best Hyperparameters: {'metric': 'euclidean', 'n_neighbors': 9}
Accuracy: 0.9948411060668593
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     18759
           1       0.87      0.98      0.92       625

    accuracy                           0.99     19384
   macro avg       0.94      0.99      0.96     19384
weighted avg       1.00      0.99      0.99     19384
```

0s

```python
# Get distances to nearest neighbors
distances, indices = knn.kneighbors(X_train_scaled)

# Calculate feature importance based on average distances
feature_importance = np.mean(distances, axis=0)

# Normalize feature importance scores
feature_importance /= np.sum(feature_importance)

# Print feature importance
print("Feature Importance based on KNN distances:")
for feature, importance in zip(X.columns, feature_importance):
    print(f"{feature}: {importance}")
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X do
  warnings.warn(
Feature Importance based on KNN distances:
loan_size: 0.19730636117804023
interest_rate: 0.19794736548892317
borrower_income: 0.1999896131547592
debt_to_income: 0.20205079564338377
num_of_accounts: 0.20270586453489378
```

Adaboost:

ADABOOST

```
[ ] # Initialize AdaBoostClassifier
    ada_model = AdaBoostClassifier(n_estimators=100, random_state=42)
```

```
[ ] # Fit the model to the training data
    ada_model.fit(X_train, y_train)
```

```
▾ AdaBoostClassifier
AdaBoostClassifier(n_estimators=100, random_state=42)
```

```
[ ] # Use feature importance for feature selection
    feature_importance = ada_model.feature_importances_
    # Select features based on importance
    selected_features_ada = SelectFromModel(ada_model, threshold='median')
    # Transform the train and test datasets
    X_train_selected = selected_features_ada.transform(X_train)
    X_test_selected = selected_features_ada.transform(X_test)

    # Print selected features
    print("Selected Features using AdaBoostClassifier:")
    selected_feature_indexes = selected_features_ada.get_support(indices=True)
    selected_features_names = X.columns[selected_feature_indexes]
    print(selected_features_names)

    print("Feature Importances after feature selection with AdaBoost:")
    for feature, importance in zip(X.columns[selected_features_ada.get_support()], ada_model.feature_importance
        print(f"{feature}: {importance}")
```

```
Selected Features using AdaBoostClassifier:
Index(['loan_size', 'interest_rate', 'debt_to_income', 'total_debt'], dtype='object')
Feature Importances after feature selection with AdaBoost:
loan_size: 0.03
interest_rate: 0.95
debt_to_income: 0.0
total_debt: 0.01
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but SelectFr
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but SelectFr
  warnings.warn(
```

```
ROC-AUC: 0.9936011514473053
```

```
# 6. Hyperparameter Tuning
# Define the hyperparameters grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.5, 1.0]
}

# Initialize AdaBoostClassifier
ada_model = AdaBoostClassifier(random_state=42)

# Initialize GridSearchCV with cross-validation
grid_search = GridSearchCV(estimator=ada_model, param_grid=param_grid, cv=5, scoring='accuracy')

# Fit the GridSearchCV to the training data using selected features
grid_search.fit(X_train_selected, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)

# 7. Evaluate Model with Best Hyperparameters
# Predict on the test set using the best model
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test_selected)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy with Best Hyperparameters:", accuracy)
```

```
Best Hyperparameters: {'learning_rate': 0.1, 'n_estimators': 50}
Accuracy with Best Hyperparameters: 0.9950990507635163
```

```
#Train AdaBoost Classifier
# Initialize AdaBoostClassifier
ada_model = AdaBoostClassifier(n_estimators=100, random_state=42)

# Fit the model to the training data
ada_model.fit(X_train_selected, y_train)

# 5. Evaluate the Model
# Make predictions on the test set
y_pred = ada_model.predict(X_test_selected)

# Calculate accuracy
accuracy_adaboost = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9917457697069748
```

```
# Assuming y_test and y_pred are your true labels and predicted labels respectively
# Calculate precision
precision = precision_score(y_test, y_pred)
print("Precision:", precision)

# Calculate recall
recall = recall_score(y_test, y_pred)
print("Recall:", recall)

# Calculate F1-score
f1 = f1_score(y_test, y_pred)
print("F1-score:", f1)

# Calculate ROC-AUC score (only applicable for binary classification)
roc_auc = roc_auc_score(y_test, y_pred)
print("ROC-AUC:", roc_auc)
```

```
Precision: 0.8732394366197183
Recall: 0.992
F1-score: 0.9288389513108614
ROC-AUC: 0.9936011514473053
```

Random Forest:

RANDOM FOREST

```
[130] # Initialize the Random Forest model
      rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
```

```
[131] # Train the model on the training data
      rf_model.fit(X_train, y_train)
```

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

```
[132] print(f"Training Score: {rf_model.score(X_train, y_train)}")
      print(f"Testing Score: {rf_model.score(X_test, y_test)}")
```

```
Training Score: 0.9973173751547668
Testing Score: 0.9917457697069748
```

```
y_true = y_test
y_pred = rf_model.predict(X_test)
print(classification_report(y_true, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     18759
           1       0.87      0.87      0.87       625

    accuracy                           0.99     19384
   macro avg       0.94      0.93      0.93     19384
weighted avg       0.99      0.99      0.99     19384
```

```
[ ] # Get feature importances from the trained Random Forest model
    feature_importances = rf_model.feature_importances_

    # Create a DataFrame to display feature importances
    feature_importance_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature_importances})

    # Sort the DataFrame by importance in descending order
    feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

    # Display the sorted feature importances
    print(feature_importance_df)
```

```
            Feature  Importance
1     interest_rate    0.315322
2   borrower_income    0.176541
0         loan_size    0.171583
6        total_debt    0.150846
3    debt_to_income    0.149850
4   num_of_accounts    0.035752
5  derogatory_marks    0.000106
```

Accuracy Comparison:

```
# Plot graph
models = ['Logistic Regression', 'AdaBoost', 'KNN', 'Random Forest']
accuracies = [accuracy_log_reg, accuracy_adaboost, accuracy_knn, accuracy_random_forest]

plt.figure(figsize=(10, 6))
plt.bar(models, accuracies, color=['blue', 'green', 'red', 'purple'])
plt.xlabel('Models')
plt.ylabel('Accuracy')
plt.title('Accuracy Comparison of Different Models')
plt.ylim(0.95, 1.0)  # Set y-axis limits for better visualization
plt.show()
```