Step 1: Load and Preprocess Data

Here, I'll assume you're using a dataset that includes features such as income, credit_history, loan_amount, and status (where status could be 1 for creditworthy and 0 for not).

```
# Import necessary libraries
Import pandas as pd
From sklearn.model_selection import train_test_split
From sklearn.preprocessing import StandardScaler, LabelEncoder
From sklearn.ensemble import RandomForestClassifier
From sklearn.metrics import accuracy_score, classification_report, confusion_matrix
Import matplotlib.pyplot as plt
Import seaborn as sns

# Load dataset (example dataset path)
Data = pd.read_csv('credit_data.csv')

# Display first few rows
Print(data.head())

# Handle missing values (you can choose imputation or dropping)
Data = data.dropna()

# Label Encoding for categorical features (if any)
Label_encoder = LabelEncoder()
Data['employment_status'] = label_encoder.fit_transform(data['employment_status'])
```

Feature scaling for numeric features

Scaler = StandardScaler()

Numeric_features = [‘income’, ‘loan_amount’, ‘credit_history’]

Data[numeric_features] = scaler.fit_transform(data[numeric_features])

Split data into features and target

X = data.drop(‘status’, axis=1) # Features

Y = data[‘status’] # Target variable

Step 2: Train-Test Split

Next, divide the data into training and testing sets.

Split the data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

Step 3: Model Selection

We’ll use a RandomForestClassifier, a strong baseline algorithm for classification tasks. You can experiment with others like LogisticRegression, SVM, or GradientBoosting.

Initialize the RandomForestClassifier

Rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

Train the model

Rf_classifier.fit(X_train, y_train)

Step 4: Model Evaluation

After training the model, evaluate its performance on the test set.

Predict on the test set

Y_pred = rf_classifier.predict(X_test)

Calculate accuracy

Accuracy = accuracy_score(y_test, y_pred)

Print(f”Accuracy: {accuracy * 100:.2f}%”)

Detailed classification report

Print(classification_report(y_test, y_pred))

Confusion matrix

Cm = confusion_matrix(y_test, y_pred)

Sns.heatmap(cm, annot=True, fmt=’d’, cmap=’Blues’, xticklabels=[‘Not Creditworthy’, ‘Creditworthy’], yticklabels=[‘Not Creditworthy’, ‘Creditworthy’])

Plt.title(‘Confusion Matrix’)

Plt.show()

Step 5: Hyperparameter Tuning (Optional)

You can fine-tune your model’s hyperparameters using GridSearchCV to improve performance.

```
From sklearn.model_selection import GridSearchCV

# Hyperparameter grid for Random Forest
Param_grid = {
  'n_estimators': [100, 200],
  'max_depth': [10, 20, None],
  'min_samples_split': [2, 5],
}

# GridSearchCV to find the best parameters
Grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=5)
Grid_search.fit(X_train, y_train)

# Best parameters and best score
Print(f"Best Params: {grid_search.best_params_}")
Print(f"Best Score: {grid_search.best_score_}")
```

Step 6: Conclusion and Model Assessment

1. Accuracy Score: The percentage of correct predictions.

2. Classification Report: Precision, recall, and F1-score for each class.

3. Confusion Matrix: Helps visualize the performance of the model.

Step 7: Model Deployment (Optional)

Once you're satisfied with the model's accuracy, you can save and deploy it.

Import joblib

Save the model

Joblib.dump(rf_classifier, 'credit_scoring_model.pkl')

Summary of Key Steps:

Data Preprocessing: Handle missing values, scale numeric features, and encode categorical ones.

Model Training: Split the data, train using Random Forest, and evaluate the model.

Performance Metrics: Accuracy, precision, recall, F1-score, and confusion matrix.

Hyperparameter Tuning: Optimize the model with GridSearchCV.