**Customer Credit Default Prediction - Final Report**

Authors:

* Anugrah Rastogi
* Dhrub Satyam
* Malleshan D

**Abstract**

This report details a machine learning–based approach for predicting customer credit default. It explores real-world financial data enriched with behavioral and demographic attributes. After thorough data preparation and exploratory analysis, we evaluated four classification models—Logistic Regression, Decision Tree, Random Forest, and XGBoost—based on ROC AUC, F1 score, precision, and recall. The Random Forest model delivered the best overall performance, balancing prediction strength with interpretability. We recommend deploying it with regular monitoring and fairness checks.

**Table of Contents**

1. Introduction
2. Data Cleaning & Preparation
3. Exploratory Data Analysis
4. Model Selection
5. Model Analysis
6. Conclusion & Recommendations
7. Appendix with notebook output

# 1. Introduction

Predicting whether a customer will default on a credit payment isn’t just a theoretical puzzle—it’s a pressing issue for financial institutions. The ability to identify risky borrowers before issuing credit can significantly reduce financial losses, streamline approval workflows, and enhance regulatory compliance.

This project, undertaken as part of the AAI500 course at the University of San Diego, leverages machine learning techniques on a real dataset containing both behavioral and demographic factors. Our core goals were:

* To develop robust predictive models for identifying likely defaulters.
* To compare multiple machine learning algorithms for accuracy and usability.
* To derive actionable insights that can inform credit evaluation strategies.

# 2. Data Cleaning / Preparation

Data preparation is one of the most important steps in any machine learning project. Clean and well-processed data helps ensure that the models we train can learn effectively and generalize to unseen data. Before modeling, we ensured the data was clean, consistent, and representative.

**Dataset Overview:**

* Training Set: 70,000 records with 82 features
* Test Set: 33,000 records with 80 features
* Target Column: The target column indicates whether a customer has defaulted (1) or not (0)

**Cleaning and Preprocessing Steps**

* Handling Missing Values: Missing values in numerical columns were replaced using median imputation. The median is preferred over the mean because it is robust to outliers, ensuring that extreme values do not skew the data.
  + Example: If the income column had missing entries, they were filled with the median income value of the entire column.
* Categorical Encoding: Categorical columns (such as boolean or string values) were encoded using Label Encoding, which converts each unique category into a number. This was applied to both training and test sets. To avoid any mismatch, encoders were fitted on the combined unique values from both datasets.
* Example: Gender: {'Male' → 0, 'Female' → 1}
* Feature Scaling: We used RobustScaler to scale numerical features. Unlike standard scaling methods, RobustScaler uses the interquartile range (IQR), which makes it highly effective in the presence of outliers.
* Benefit: Models are not overly influenced by extreme values.
* Feature Alignment: To prevent inconsistencies during prediction, we made sure the training and test datasets had the same columns. This included: Dropping extra columns if present in either set. Ensuring consistent column order.
* Target Isolation: The column target was isolated as the prediction label. 0 → No Default, 1 → Default.

# 3. Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) plays a critical role in understanding the dataset’s structure, identifying patterns, detecting anomalies, and shaping strategies for feature engineering. Our goal was to uncover relationships between input features and the target variable (default), helping guide model design.

**Univariate Analysis**

We began by examining distributions of key features to understand their spread, outliers, and potential impact:

* **Numerical Features:** age, debt\_ratio, monthly\_income, number\_of\_open\_credit\_lines, etc. Some variables showed right-skewed distributions, especially financial ones like debt\_ratio and monthly\_income.
* **Categorical Features:** Boolean and object-type columns were label-encoded. Certain categorical features had imbalanced value counts, which might influence the model bias if not handled properly.
* **Example Chart:** Histogram of debt\_ratio. Revealed a long tail indicating a small number of customers with very high debt ratios.

**Bivariate Analysis**

We analyzed how features related to the target variable.

* **Boxplots:** debt\_ratio and number\_of\_dependents were higher on average among defaulters.
* **Correlation Matrix:** Heatmaps helped spot weak to moderate correlations. Most features showed low linear correlation, suggesting nonlinear models (e.g., trees) might perform better.
* **Default Distribution:** Only ~20.6% of records were labeled as defaulters. This class imbalance required attention during model training (e.g., via class\_weight='balanced' or resampling techniques).

**Key Visual Insights**

* **Target Imbalance:** Defaulted (1): ~20% and Non-defaulted (0): ~80%. This significant imbalance influenced our choice of metrics (F1-score, ROC AUC instead of accuracy).
* **Outliers:** Features like monthly\_income had extreme values. Instead of dropping, we used RobustScaler to mitigate their influence.
* **Missing Data:** Found in income and dependent count fields. Imputed using median values, preserving the distribution shape.

**Summary of EDA Outcomes**

|  |  |
| --- | --- |
| **Insight** | **Action Taken** |
| Class imbalance in target | Used class\_weight='balanced' and chose suitable metrics |
| Skewed financial data | Applied RobustScaler |
| Categorical encoding needed | Used LabelEncoder |
| Correlations were weak | Preferred ensemble models like Random Forest/XGBoost |
| Missing values in key columns | Used median imputation |

# 4. Model Selection

Choosing the right model is essential when dealing with real-world data—especially one that involves imbalanced classes like credit default prediction. Our selection process involved testing multiple algorithms with varying complexity and interpretability.

**Candidate Models**

We explored four supervised classification algorithms, each with its unique advantages:

|  |  |
| --- | --- |
| **Model** | **Rationale** |
| Logistic Regression | Simple, interpretable, works well as a baseline. |
| Decision Tree | Easy to visualize and explain decisions. |
| Random Forest | Handles non-linearities, robust to noise, good for tabular data. |
| XGBoost | State-of-the-art gradient boosting algorithm with excellent accuracy and performance. |

**Hyperparameter Tuning**

Each model was fine-tuned using GridSearchCV with 5-fold cross-validation to optimize for ROC AUC:

**Logistic Regression:**

* Grid: C values and solvers
* class\_weight='balanced' was used to address the target imbalance

**Decision Tree:**

* Grid: max\_depth, min\_samples\_split, min\_samples\_leaf

**Random Forest:**

* Grid: n\_estimators, max\_depth, min\_samples\_split

**XGBoost:**

* Grid: n\_estimators, max\_depth, learning\_rate
* Used eval\_metric='logloss'

**Tools Used:**

* GridSearchCV for tuning
* RobustScaler to scale numerical features
* LabelEncoder for categorical encoding

**Evaluation Strategy**

To evaluate models fairly, we split the training data:

* **80% for training**, **20% for validation**
* Performance was measured using:
  + **ROC AUC Score** (discrimination ability)
  + **F1-Score** (balance between precision and recall)
  + **Precision & Recall** (for defaulters)

**Validation Results:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **ROC AUC** | **F1 Score** | **Precision** | **Recall** |
| Logistic Regression | 0.9057 | 0.7134 | 0.63 | 0.82 |
| Decision Tree | 0.8808 | 0.6855 | 0.75 | 0.63 |
| Random Forest | 0.9633 | 0.8049 | 0.85 | 0.76 |
| XGBoost | 0.9576 | 0.7906 | 0.82 | 0.76 |

**Best Model Selected**

After comparing all results: Random Forest was selected as the final model due to its highest ROC AUC, strong F1-score, and balanced precision-recall trade-off.

# 5. Model Analysis

After training and tuning, we analyzed each model’s performance on the validation set to assess its real-world applicability. The focus was on evaluating how well each model could identify defaulters (class 1) without generating too many false positives.

**Classification Reports**

Each model’s performance was evaluated using precision, recall, and F1-score, focusing especially on class 1 (default cases):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Model** | **Precision (1)** | **Recall (1)** | **F1 Score (1)** | **ROC AUC** |
| Logistic Regression | 0.63 | 0.82 | 0.7134 | 0.9057 |
| Decision Tree | 0.75 | 0.63 | 0.6855 | 0.8808 |
| Random Forest | 0.85 | 0.76 | 0.8049 | 0.9633 |
| XGBoost | 0.82 | 0.76 | 0.7906 | 0.9576 |

**Interpretation of Metrics:**

* Precision indicates how many of the predicted defaults were actually correct.
* Recall shows how many actual defaults were correctly identified.
* F1 Score balances both precision and recall.
* ROC AUC reflects the overall ability to distinguish between classes.

Random Forest led with the highest F1 and ROC AUC, making it best suited for production.

**Visual Insights:**

We plotted key evaluation curves to support our findings:

1. ROC Curve – Random Forest leads with the highest AUC

2. Precision-Recall Curve – Confirms the effectiveness in handling class imbalance

3. Model Comparison Bar Chart

|  |  |  |
| --- | --- | --- |
| **Model** | **ROC AUC** | **F1 Score** |
| Random Forest | 0.9633 | 0.8049 |
| XGBoost | 0.9576 | 0.7906 |
| Logistic Reg. | 0.9057 | 0.7134 |
| Decision Tree | 0.8808 | 0.6855 |

**Summary of Observations**

* Random Forest performed best overall, especially in identifying defaults accurately.
* XGBoost was a close second and could be explored further for optimization.
* Logistic Regression, while simple, showed surprisingly good recall but low precision—making it risky for production.
* Decision Tree was interpretable but too simplistic for this complex dataset.

**Why Random Forest?**

* Superior ROC AUC and F1-Score.
* Balanced performance on both classes.
* Robust to outliers and feature noise.
* Suitable for financial datasets with mixed feature types.

# 6. Conclusion and Recommendations

**Project Recap:**

This project aimed to build a machine learning solution to predict customer credit default using real-world tabular data. Through systematic preprocessing, model building, and evaluation, we were able to explore multiple classification models and identify the most effective solution.

**Key Findings:**

Random Forest emerged as the most reliable model, outperforming others with:

* Highest ROC AUC (0.9633)
* Strong F1 Score (0.8049)
* Balanced precision (0.85) and recall (0.76)

XGBoost showed competitive results, slightly behind Random Forest, and can be considered for further tuning or ensemble techniques. Logistic Regression, though basic, demonstrated strong recall (0.82), meaning it rarely missed defaulters, but suffered from low precision (0.63), potentially flagging too many false positives. Decision Tree underperformed, indicating a need for deeper tree-based models or ensemble methods for such complex data.

**Business Implications:**

A robust credit default prediction model offers banks and financial institutions the ability to: Minimize financial losses by identifying risky customers early, Enhance customer experience by streamlining approvals for low-risk applicants, Comply with regulatory requirements using explainable and auditable models, and Reduce operational costs by automating part of the decision-making process.

**Recommendations**

To further improve model performance and business adoption:

* Feature Engineering: Create features like trend indicators, rolling averages, or recent transaction history. Explore time-window aggregation for behavior-based features.
* Model Optimization: Perform more extensive hyperparameter tuning (e.g., using randomized search). Explore stacking or voting ensembles of top models (e.g., Random Forest + XGBoost).
* Bias & Fairness Audits: Evaluate whether the model shows bias against any demographic group. Perform fairness testing to ensure ethical compliance.
* Model Monitoring: Set up a feedback loop to monitor predictions and update the model regularly. Use A/B testing to evaluate model performance against existing systems.
* Interpretability: Apply SHAP (SHapley Additive exPlanations) or LIME for model explainability. Provide justification for each prediction in customer-facing applications.

**Final Recommendation**

Based on current evaluation metrics, Random Forest is recommended for deployment as the primary model. However, continual monitoring, feedback integration, and further experimentation with advanced techniques will help refine predictions in real-world environments.

Appendix (Code and Outputs)

**Appendix A.1 Load Data**

**Code:**

*#Importing Libraries*

**import** pandas **as** pd

**import** numpy **as** np

**import** os

**from** sklearn.model\_selection **import** train\_test\_split, GridSearchCV

**from** sklearn.preprocessing **import** RobustScaler, LabelEncoder

**from** sklearn.ensemble **import** RandomForestClassifier

**from** sklearn.tree **import** DecisionTreeClassifier

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.metrics **import** classification\_report, roc\_auc\_score, f1\_score

**from** xgboost **import** XGBClassifier

**import** matplotlib.pyplot **as** plt

**%matplotlib** inline

**import** warnings

warnings**.**filterwarnings('ignore')

*# Check current working directory and file existence*

print(f"Current working directory: {os**.**getcwd()}")

train\_file **=** "processed\_train\_data.csv"

test\_file **=** "processed\_test\_data.csv"

**if** **not** os**.**path**.**exists(train\_file) **or** **not** os**.**path**.**exists(test\_file):

**raise** FileNotFoundError(f"One or both files not found: {train\_file}, {test\_file}")

*# Load datasets*

train\_df **=** pd**.**read\_csv(train\_file)

test\_df **=** pd**.**read\_csv(test\_file)

print(f"Train data shape: {train\_df**.**shape}")

print(f"Test data shape: {test\_df**.**shape}")

**Appendix A.2 Preprocess Data (Data Cleaning)**

**Code:**

*# Define target column*

target\_col **=** 'target'

**if** target\_col **not** **in** train\_df**.**columns:

**raise** ValueError(f"Target column '{target\_col}' not found in training data")

*# Separate features and target*

X\_train\_full **=** train\_df**.**drop(columns**=**[target\_col])

y\_train\_full **=** train\_df[target\_col]

X\_test\_final **=** test\_df**.**drop(columns**=**[target\_col], errors**=**'ignore')

*# Align train and test columns*

common\_cols **=** X\_train\_full**.**columns**.**intersection(X\_test\_final**.**columns)

X\_train\_full **=** X\_train\_full[common\_cols]

X\_test\_final **=** X\_test\_final[common\_cols]

print(f"Aligned columns: {len(common\_cols)}")

*# Handle missing values*

numeric\_cols **=** X\_train\_full**.**select\_dtypes(include**=**[np**.**number])**.**columns

**for** col **in** numeric\_cols:

median\_val **=** X\_train\_full[col]**.**median()

X\_train\_full[col]**.**fillna(median\_val, inplace**=True**)

X\_test\_final[col]**.**fillna(median\_val, inplace**=True**)

*# Encode categorical columns*

label\_encoders **=** {}

categorical\_cols **=** X\_train\_full**.**select\_dtypes(include**=**['object', 'bool'])**.**columns

**for** col **in** categorical\_cols:

le **=** LabelEncoder()

*# Fit on combined train and test unique values to avoid unseen labels*

unique\_vals **=** pd**.**concat([X\_train\_full[col], X\_test\_final[col]], axis**=**0)**.**astype(str)**.**unique()

le**.**fit(unique\_vals)

X\_train\_full[col] **=** le**.**transform(X\_train\_full[col]**.**astype(str))

X\_test\_final[col] **=** le**.**transform(X\_test\_final[col]**.**astype(str))

label\_encoders[col] **=** le

*# Split training data into train and validation sets*

X\_train, X\_val, y\_train, y\_val **=** train\_test\_split(

X\_train\_full, y\_train\_full, test\_size**=**0.2, random\_state**=**42, stratify**=**y\_train\_full

)

*# Scale features*

scaler **=** RobustScaler()

X\_train\_scaled **=** scaler**.**fit\_transform(X\_train)

X\_val\_scaled **=** scaler**.**transform(X\_val)

X\_test\_scaled **=** scaler**.**transform(X\_test\_final)

print(f"Training set: {X\_train**.**shape}, Validation set: {X\_val**.**shape}, Test set: {X\_test\_final**.**shape}")

**Appendix A.3 Train Models with Hyperparameter Tuning**

**Code:**

*# Initialize models and parameter grids*

models **=** {

'RandomForest': {

'model': RandomForestClassifier(random\_state**=**42, n\_jobs**=-**1),

'param\_grid': {

'n\_estimators': [50, 100],

'max\_depth': [10, 20, **None**],

'min\_samples\_split': [2, 5]

}

},

'DecisionTree': {

'model': DecisionTreeClassifier(random\_state**=**42),

'param\_grid': {

'max\_depth': [5, 10, 15],

'min\_samples\_split': [2, 5],

'min\_samples\_leaf': [1, 2]

}

},

'XGBoost': {

'model': XGBClassifier(random\_state**=**42, n\_jobs**=-**1, eval\_metric**=**'logloss'),

'param\_grid': {

'n\_estimators': [50, 100],

'max\_depth': [3, 6],

'learning\_rate': [0.1, 0.01]

}

},

'LogisticRegression': {

'model': LogisticRegression(random\_state**=**42, max\_iter**=**10000, class\_weight**=**'balanced'),

'param\_grid': {

'C': [0.1, 1.0, 10.0],

'solver': ['lbfgs', 'liblinear']

}

}

}

*# Train and tune models*

best\_models **=** {}

**for** name, config **in** models**.**items():

print(f"\nTuning {name}...")

grid\_search **=** GridSearchCV(

config['model'], config['param\_grid'], cv**=**5, scoring**=**'roc\_auc', n\_jobs**=-**1

)

grid\_search**.**fit(X\_train\_scaled, y\_train)

best\_models[name] **=** grid\_search**.**best\_estimator\_

print(f"Best parameters: {grid\_search**.**best\_params\_}")

print(f"Best cross-validation ROC-AUC: {grid\_search**.**best\_score\_:.4f}")

This code block performs the following:

* Splits the training data into training and validation sets.
* Applies a grid search for tuning hyperparameters (n\_estimators, max\_depth, min\_samples\_split).
* Evaluates the best model based on ROC AUC and F1 score.
* Outputs a classification report.

**Output:**

**Tuning RandomForest**

* Best parameters: {'max\_depth': None, 'min\_samples\_split': 2, 'n\_estimators': 100}
* Best cross-validation ROC-AUC: 0.9559

**Tuning DecisionTree**

* Best parameters: {'max\_depth': 10, 'min\_samples\_leaf': 2, 'min\_samples\_split': 5}
* Best cross-validation ROC-AUC: 0.8708

**Tuning XGBoost**

* Best parameters: {'learning\_rate': 0.1, 'max\_depth': 6, 'n\_estimators': 100}
* Best cross-validation ROC-AUC: 0.9536

**Tuning LogisticRegression**

* Best parameters: {'C': 0.1, 'solver': 'liblinear'}
* Best cross-validation ROC-AUC: 0.9074

**Appendix A.4: Model Evaluation**

Evaluated all models on the validation set using ROC-AUC, F1-score, and classification report using the following code.

**Code:**

*# Evaluate each model*

results = {}

for name, model in best\_models.items():

print(f"\n{name} Validation Results:")

y\_val\_pred = model.predict(X\_val\_scaled)

y\_val\_proba = model.predict\_proba(X\_val\_scaled)[:, 1]

results[name] = {

'roc\_auc': roc\_auc\_score(y\_val, y\_val\_proba),

'f1\_score': f1\_score(y\_val, y\_val\_pred),

'y\_val\_pred': y\_val\_pred,

'y\_val\_proba': y\_val\_proba

}

print(classification\_report(y\_val, y\_val\_pred))

print(f"ROC AUC: {results[name]['roc\_auc']:.4f}")

print(f"F1 Score: {results[name]['f1\_score']:.4f}")

**Output:**

Below is the detailed classification report, including precision, recall, F1-score, and support:

**RandomForest Validation Results:**

precision recall f1-score support

0 0.92 0.96 0.94 10471

1 0.85 0.76 0.80 3529

accuracy 0.91 14000

macro avg 0.89 0.86 0.87 14000

weighted avg 0.91 0.91 0.91 14000

ROC AUC: 0.9633

F1 Score: 0.8049

**DecisionTree Validation Results:**

precision recall f1-score support

0 0.88 0.93 0.91 10471

1 0.75 0.63 0.69 3529

accuracy 0.85 14000

macro avg 0.82 0.78 0.80 14000

weighted avg 0.85 0.85 0.85 14000

ROC AUC: 0.8808

F1 Score: 0.6855

**XGBoost Validation Results:**

precision recall f1-score support

0 0.92 0.94 0.93 10471

1 0.82 0.76 0.79 3529

accuracy 0.90 14000

macro avg 0.87 0.85 0.86 14000

weighted avg 0.90 0.90 0.90 14000

ROC AUC: 0.9576

F1 Score: 0.7906

**LogisticRegression Validation Results:**

precision recall f1-score support

0 0.93 0.84 0.88 10471

1 0.63 0.82 0.71 3529

accuracy 0.83 14000

macro avg 0.78 0.83 0.80 14000

weighted avg 0.86 0.83 0.84 14000

ROC AUC: 0.9057

F1 Score: 0.7134

**Model Comparison Table**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Accuracy** | **ROC AUC** | **F1 Score**  **(Class 1)** | **Precision**  **(Class 1)** | **Recall**  **(Class 1)** |
| Random Forest | 0.91 | 0.9633 | 0.8049 | 0.85 | 0.76 |
| XGBoost | 0.90 | 0.9576 | 0.7906 | 0.82 | 0.76 |
| Decision Tree | 0.85 | 0.8808 | 0.6855 | 0.75 | 0.63 |
| Logistic Regression | 0.83 | 0.9057 | 0.7134 | 0.63 | 0.82 |

**Appendix A.5: Model Performance Comparison**

To compare the effectiveness of all trained models, we evaluated them using the validation dataset based on key performance metrics: Accuracy, ROC AUC, F1-Score, Precision, and Recall.

We have created a grouped bar chart comparing ROC AUC and F1-Score for each model using the following code.

**Code:**

*# Collect metrics*

model\_names **=** list(results**.**keys())

roc\_auc\_scores **=** [results[name]['roc\_auc'] **for** name **in** model\_names]

f1\_scores **=** [results[name]['f1\_score'] **for** name **in** model\_names]

*# Plot comparison*

fig, ax **=** plt**.**subplots(figsize**=**(10, 6))

x **=** np**.**arange(len(model\_names))

width **=** 0.35

ax**.**bar(x **-** width**/**2, roc\_auc\_scores, width, label**=**'ROC AUC', color**=**'skyblue')

ax**.**bar(x **+** width**/**2, f1\_scores, width, label**=**'F1 Score', color**=**'lightcoral')

ax**.**set\_xlabel('Models')

ax**.**set\_ylabel('Scores')

ax**.**set\_title('Model Performance Comparison')

ax**.**set\_xticks(x)

ax**.**set\_xticklabels(model\_names, rotation**=**45)

ax**.**legend()

**for** i, (roc, f1) **in** enumerate(zip(roc\_auc\_scores, f1\_scores)):

ax**.**text(i **-** width**/**2, roc **+** 0.01, f'{roc:.3f}', ha**=**'center')

ax**.**text(i **+** width**/**2, f1 **+** 0.01, f'{f1:.3f}', ha**=**'center')

plt**.**tight\_layout()

plt**.**show()

**Output:**

A graph of different colored bars

AI-generated content may be incorrect.

**Appendix A.6: Make Final Predictions**

Select the best model based on ROC-AUC and predict on the test set.# Select best model based on ROC-AUC using the following code.

**Code:**

best\_model\_name = max(results, key=lambda x: results[x]['roc\_auc'])

best\_model = best\_models[best\_model\_name]

print(f"Best model: {best\_model\_name} (ROC AUC: {results[best\_model\_name]['roc\_auc']:.4f})")

*# Make predictions*

predictions = best\_model.predict(X\_test\_scaled)

probabilities = best\_model.predict\_proba(X\_test\_scaled)[:, 1]

print(f"Generated {len(predictions)} predictions")

print(f"Default rate: {(predictions == 1).mean():.4f}")

*# Save predictions*

output = pd.DataFrame({'Prediction': predictions, 'Probability': probabilities})

output.to\_csv('test\_predictions.csv', index=False)

print("Predictions saved to 'test\_predictions.csv'")

**Output:**

Best model: RandomForest (ROC AUC: 0.9633)

Generated 33000 predictions

Default rate: 0.2061

Predictions saved to 'test\_predictions.csv'

**Appendix A.7: Visualizations**

To support the model evaluation and comparison, we generated the following visualizations:

1. **ROC Curve – Receiver Operating Characteristic:** This curve helps to visualize the trade-off between the True Positive Rate (Recall) and the False Positive Rate for different classification thresholds.
2. **Precision-Recall Curve:** This curve is especially useful for imbalanced datasets, as it focuses on the minority class (defaults in our case).

The following code is used to generate ROC Curve and Precision-Recall Curve.

**Code:**

**import** matplotlib.pyplot **as** plt

**import** seaborn **as** sns *# Required for heatmap; ensure seaborn is installed*

**from** sklearn.metrics **import** roc\_curve, precision\_recall\_curve, auc

**import** numpy **as** np

**import** pandas **as** pd

*# Set up the plotting style (use 'ggplot' or another valid matplotlib style if seaborn is unavailable)*

plt**.**style**.**use('ggplot') *# Fallback style; alternatively, use 'seaborn-v0\_8' if seaborn is installed*

*# Create a figure with subplots*

fig **=** plt**.**figure(figsize**=**(15, 12))

*# 1. ROC Curve*

plt**.**subplot(2, 2, 1)

**for** name, model **in** best\_models**.**items():

y\_val\_proba **=** model**.**predict\_proba(X\_val\_scaled)[:, 1]

fpr, tpr, \_ **=** roc\_curve(y\_val, y\_val\_proba)

roc\_auc **=** auc(fpr, tpr)

plt**.**plot(fpr, tpr, label**=**f'{name} (AUC = {roc\_auc:.3f})')

plt**.**plot([0, 1], [0, 1], 'k--', label**=**'Random')

plt**.**xlabel('False Positive Rate')

plt**.**ylabel('True Positive Rate')

plt**.**title('ROC Curves')

plt**.**legend(loc**=**'lower right')

plt**.**grid(**True**)

*# 2. Precision-Recall Curve*

plt**.**subplot(2, 2, 2)

**for** name, model **in** best\_models**.**items():

y\_val\_proba **=** model**.**predict\_proba(X\_val\_scaled)[:, 1]

precision, recall, \_ **=** precision\_recall\_curve(y\_val, y\_val\_proba)

pr\_auc **=** auc(recall, precision)

plt**.**plot(recall, precision, label**=**f'{name} (AUC = {pr\_auc:.3f})')

plt**.**xlabel('Recall')

plt**.**ylabel('Precision')

plt**.**title('Precision-Recall Curves')

plt**.**legend(loc**=**'lower left')

plt**.**grid(**True**)

*# # Display all plots*

*# plt.suptitle('Model Evaluation Plots', fontsize=16, y=1.05)*

plt**.**show()

**Output:**

A comparison of graphs and charts

AI-generated content may be incorrect.

These visualizations support data-driven decisions and reinforce the selection of Random Forest as the final model based on balanced performance across multiple metrics.