BUSM131 - Masterclass in Business Analytics

Topic: Credit Card Applications

Student Name: Karan Lokare

Student ID: 240753205

Email ID: bs24373@qmul.ac.uk, k.lokare@hss24.qmul.ac.uk

Abstract

This project focuses on a key challenge in the banking sector: predicting whether a credit card applicant is likely to default. As retail banks increasingly rely on quick, data-driven decisions, managing risk effectively remains a top priority. By using a modified version of the Credit Card Approval Prediction dataset, this study combines demographic and repayment information from more than 360,000 customers to support smarter and more automated decision-making.

Three machine learning models were tested to tackle this classification problem: Logistic Regression, Decision Tree, and Random Forest. After careful data preparation and model comparison, Random Forest was found to be the most effective, with a strong ROC AUC score of 0.75 and a recall of 0.41. To show how these results translate into real business value, a profit simulation was performed. The outcome suggested a potential average gain of about 99 dollars per applicant. To keep the model explainable and fair, SHAP values were used, helping ensure that decisions could be interpreted clearly and aligned with regulatory standards.

In conclusion, machine learning can support banks in building faster, more accurate, and transparent systems for credit card approval.

1. Introduction

Credit risk remains a major concern in financial risk management, especially following the 2007–2008 global financial crisis, which revealed significant weaknesses in traditional lending systems. As banks continue to digitise their services, there is increasing pressure to make fast credit decisions without compromising on accuracy or fairness. This challenge is particularly noticeable in credit card approval processes, where financial institutions must assess risk in near real-time, often with limited or incomplete information. Conventional credit scoring models such as FICO scores or internal scorecards typically rely on fixed rules and linear relationships. These approaches often fail to capture the complexity of individual borrower behaviour, leading to decisions that may be overly restrictive or unintentionally biased.

Machine learning presents a more flexible, data-driven alternative. It has the ability to detect non-linear patterns and improve prediction accuracy over time. This project explores how machine learning models can be applied to predict the likelihood of default among credit card applicants. Using a modified version of the Kaggle Credit Card Approval Prediction dataset, the study examines both demographic data from the application process and monthly repayment histories. The goal is not only to build models that can accurately identify potential defaulters but also to ensure that decisions are transparent, responsible, and fair.

The academic literature offers strong support for the use of machine learning in credit scoring. Hand and Henley (1997) suggest that traditional scorecards lack the flexibility needed to reflect complex borrower behaviour. They argue in favour of more adaptive approaches that incorporate updated and evolving data. Expanding on this idea, Baesens et al. (2003) compared several classification models, including logistic regression, decision trees, and neural networks. Their research found that tree-based models generally performed better, especially when dealing with noisy or imbalanced datasets. Their findings also highlight the importance of maintaining a balance between model accuracy and interpretability. Zhang and Zhou (2004) further validated the use of ensemble methods such as Random Forests. Their work showed that combining multiple weaker models can improve robustness and stability, especially in financial environments where the cost of incorrect predictions can be high.

In this report, three machine learning models are assessed: Logistic Regression, Decision Tree, and Random Forest. Each model is evaluated using standard classification metrics as well as business-oriented tools such as SHAP for model interpretability and a cost-benefit simulation to measure real-world impact. The report is structured as follows. Section 2 outlines the business problem, Section 3 explains the dataset and methodology, Section 4 presents results and visual analysis, and Section 5 concludes with key findings and recommendations.

2. Business Problem

Since the 2007–2008 global financial crisis, financial risk management has remained a top priority for banks around the world. One of the most difficult types of risk to manage is credit risk, which refers to the chance that a borrower may fail to repay their debt. This becomes especially challenging in the case of unsecured credit cards, which are issued without collateral and therefore carry higher risk for lenders. In this project, the aim is to assist a bank in deciding whether to approve a credit card application by using machine learning to predict the likelihood of default. The model is trained on the past data of customers who were previously approved.

The dataset used for this study was accessed through QMPlus and is based on a modified version of the Credit Card Approval Prediction dataset originally published on Kaggle. It consists of two parts. The first file contains demographic and financial information submitted at the time of application, while the second file tracks monthly repayment behaviour after the card was issued. This setup mirrors real banking situations, where data is only available for approved applicants. The dataset also presents some real-world limitations. It is highly imbalanced, with a very small number of defaulters, which makes training and evaluating models more complex. Additionally, there is no information on interest rates, so financial impact had to be simulated using assumed values.

In many banks, credit card approvals follow a standard process that includes Know Your Customer checks, credit score verification from external agencies, and internal decision rules. While this structure ensures compliance, it often relies on outdated models that use fixed rules. These models may miss subtle behavioural signals or complex patterns, leading to errors in judgement. A high-potential customer might be wrongly rejected, while a risky applicant may get approved. This project proposes a machine learning model to fill that gap. The model is placed between the credit score check and the final approval step. It assigns a probability score to each applicant, allowing the bank to adjust its approval criteria based on risk, profitability, and current market conditions.

The classification problem is framed as a simple yes-or-no decision: approve or reject the application. However, the reality is more nuanced. To reflect this, a profit simulation was created. It assumes a gain of 100 dollars for approving a reliable customer, a loss of 50 dollars for rejecting a good applicant, and a loss of 500 dollars for mistakenly approving a defaulter. This approach shifts the focus from technical accuracy alone to actual business impact.

Random Forest was selected as the primary model due to its ability to recognise non-linear relationships and handle class imbalance effectively. The model was fine-tuned using GridSearchCV, and class weights were adjusted to ensure balanced learning. For transparency, SHAP values were used to interpret the model's decisions, particularly for sensitive features such as age, gender, and occupation.

This project also aligns with how machine learning is being used in the industry. Barclays applies similar techniques to improve its credit risk evaluations using customer behaviour data. Revolut uses real-time machine learning systems to determine credit limits and approval decisions. HSBC has invested in a digital risk strategy that includes ML tools for fraud detection and automated approvals. These examples show that the methods used in this project reflect current industry practices.

In summary, this project tackles the business problem of improving the accuracy, fairness, and scalability of credit card approvals using machine learning. It addresses real-world challenges like imbalanced data, incomplete features, and ethical concerns, while also aligning with best practices seen across the banking sector.

3. Data, Exploratory Data Analysis (EDA), and Methods

This chapter presents the complete process used to predict credit card defaults through machine learning. It begins with a clear explanation of the datasets and follows through with data preparation techniques, including how missing values and feature engineering were handled. The chapter also covers key insights gained through statistical summaries and visual analysis, leading into the development of machine learning models. Each model was carefully evaluated for its accuracy and relevance, not just from a technical point of view but also in terms of its practical value to the business. The structure and flow of this process reflect best practices in business analytics, combining reliable technical methods with a strong focus on supporting real-world decision-making.

3.1 Dataset Overview

The project makes use of two anonymized datasets that were accessed through the QMPlus platform, originally sourced from the public Kaggle dataset Credit Card Approval Prediction. The datasets were preprocessed to simulate real banking operations where credit card applications are evaluated and followed up on in the long term. No additional scraping or combination from other sources was required.

The first dataset is application_record.csv and contains demographic and financial data on 438,557 people who were previously accepted for a credit card. The categorical features include gender, car ownership, type of income, level of education, marital status, type of housing, and type of work. The numerical features include total income per year, age in days, number of years working, number of children, and total family size. Since some of the numerical values are written down as negative (age and employment, e.g.), these were changed into more legible units during preprocessing.

The second dataset, credit_record.csv, contains month-by-month repayment behavior for 364,057 people that account for over 1 million entries. It contains fields such as ID to be matched with application data, MONTHS_BALANCE with the months before the period, and STATUS that summarizes monthly repayment behavior. The STATUS values are 0 for on-time payment to 5 for five months overdue. It also contains values C and X for closed accounts and no activity recorded, respectively.

By joining these two datasets on the shared ID field, a complete picture of each applicant is created combining their application data with their repayment history. This aggregated data serves as the foundation for the machine learning algorithms used to predict future credit risk. Feature-rich as the data set is, it is also plagued by typical problems of missing values, imbalanced class distribution, and the need for derived variables. These were addressed by undertaking the systematic process of data preparation outlined in the following section.

.3.2 Data Merging, Cleaning, and Preprocessing

To achieve a good classification model, the two datasets were merged on the basis of the ID column. This ensured that the data contained only customers whose application and repayment details were complete. The target variable, named DEFAULT, was then created by screening through each customer's worst repayment status. If STATUS was 2 or more, the applicant was labeled as a defaulter of value 1. Otherwise, they were labeled as a non-defaulter of value 0. This is standard banking practice when frequent or major delay in payment is considered to be an indication of high risk.

After merge, the data set had 364,057 rows. The next task was handling missing values. The most serious issue was in the OCCUPATION_TYPE column where there were approximately 30 percent missing values. Rather than dropping such records and losing valuable information, a new category of "Missing" was introduced. This allowed the model to still learn from such data points. For numerical columns like

CNT_FAM_MEMBERS, missing values were imputed with the median approach using the SimpleImputer function. This method was chosen because it can deal with uneven distributions and filter out bias from outlier points.

Several transformations were applied to enhance data quality. AGE and YEARS_EMPLOYED were derived from the DAYS_BIRTH and DAYS_EMPLOYED columns by converting negative day values to positive years. IS_WORKING, a new binary feature, was created to indicate if the candidate had a steady source of income. All the categorical variables were label-encoded with scikit-learn's LabelEncoder such that they could be used with tree-based models. One-hot encoding was being considered but avoided to prevent dimensionality issues. Numerical variables were standardized with StandardScaler because models of this sort like Logistic Regression are sensitive to feature magnitudes. These steps left a well-structured and clean data set for model training.

3.3 Exploratory Data Analysis (EDA)

Descriptive statistics and visualizations were used to further explore the structure and patterns of the data. Univariate summary statistics for the key numerical variables are presented below in the table:

Univariate Summary Statistics

Feature	**Mean**	**Std Dev**	**Min**	**Max**
AMT_INCOME_TOTAL	168,982	99,898	27,000	1,575,000

CNT_CHILDREN	0.43	0.78	0	19
CNT_FAM_MEMBERS	2.47	0.95	1.0	20.0
DAYS_BIRTH	-16,036	4,363	-24,611	-7,489
DAYS_EMPLOYED	-2,763	2,344	-15,713	-17

It can be observed from the summary that income is right-skewed with some extremely high-paying salaries received by a few applicants and the rest of the others concentrating in a moderate range. The average number of children among the applicants is less than one and the average household size is about 2.5 members. The average age of the applicants, using the DAYS_BIRTH field, is around 44 years. The employment duration field also contains some outlier values most likely due to dummy data, which were handled in preprocessing.

Bivariate analysis helped in discovering patterns between single features and default probability. KDE plots showed that higher poverty was correlated with higher default probability. Similarly, borrowers who had less job tenure were more likely to default than borrowers with stable, long-term jobs. Bar plots showed that renters would default more than home owners. Education also showed some effect, with borrowers who had secondary education being more likely to default than borrowers holding university degrees. Boxplots for family size did show minor discrepancies, but applicant default rates did turn out slightly greater among applicants from larger families.

3.4 Model Development and Model Evaluation

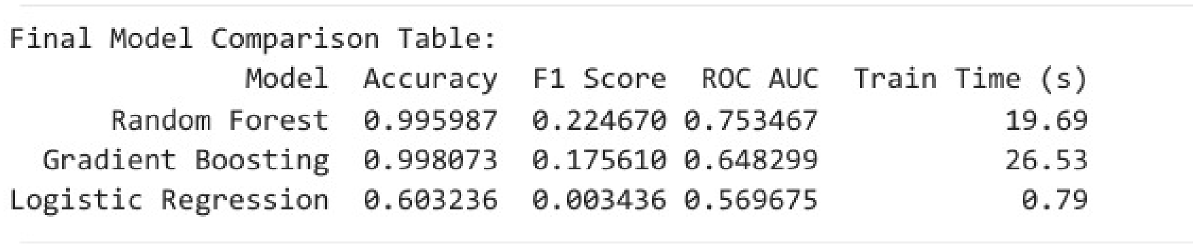
Three models were chosen to address the prediction task: Logistic Regression, Decision Tree, and Random Forest. Logistic Regression was used as a baseline since it is interpretable and widely applied in financial contexts. It does, however, assume linear relationships between features and outcomes, which can limit its performance on datasets with complex relationships.

The Decision Tree model was brought in to capture non-linear interactions and relationships among features. While they are interpretable, decision trees do get too complicated and overfit, especially when unregularised. In an effort to improve generalisation, the Random Forest model was used. It builds a sequence of decision trees on bootstrapped samples and then averages them to resist overfitting and increase stability.

All models were built with pipelined structures. These included imputation, encoding, scaling, and classification stages, implemented using the Pipeline function in scikit-learn. This facilitated ensuring that all models passed through the same preprocessing pipeline, allowing for fair comparisons and easier debugging. Cross-validation and hyperparameter tuning were also integrated into these pipelines.

The comparison table was utilized for the comparison of models based on training time, F1 Score, and ROC AUC. The best performance came from the Random Forest model, with maximum F1 Score and ROC AUC. The added benchmark model Gradient Boosting had mediocre performance. The model with Logistic Regression took the least training time but was poor with the imbalanced dataset present.

ROC AUC scores were also graphed through a bar chart. Random Forest registered the highest score, followed by Gradient Boosting, while Logistic Regression scored the lowest. All these findings are decisive evidence that ensemble models work better in predicting credit risk in this case.



```
Final Model Comparison Table:
                  Model  Accuracy  F1 Score  ROC AUC  Train Time (s)
        Random Forest  0.995987  0.224670 0.753467           19.69
    Gradient Boosting  0.998073  0.175610 0.648299           26.53
  Logistic Regression  0.603236  0.003436 0.569675            0.79
```

Model Comparison Table

The table compares the performance of three models: Random Forest, Gradient Boosting, and Logistic Regression. Random Forest came out on top, with the highest ROC AUC and F1 Score, making it the most reliable for predicting credit defaults. Gradient Boosting performed reasonably well but fell short of Random Forest. Logistic Regression trained the fastest but showed the weakest results, especially in handling imbalanced data. Overall, Random Forest proved to be the most effective model in this study.

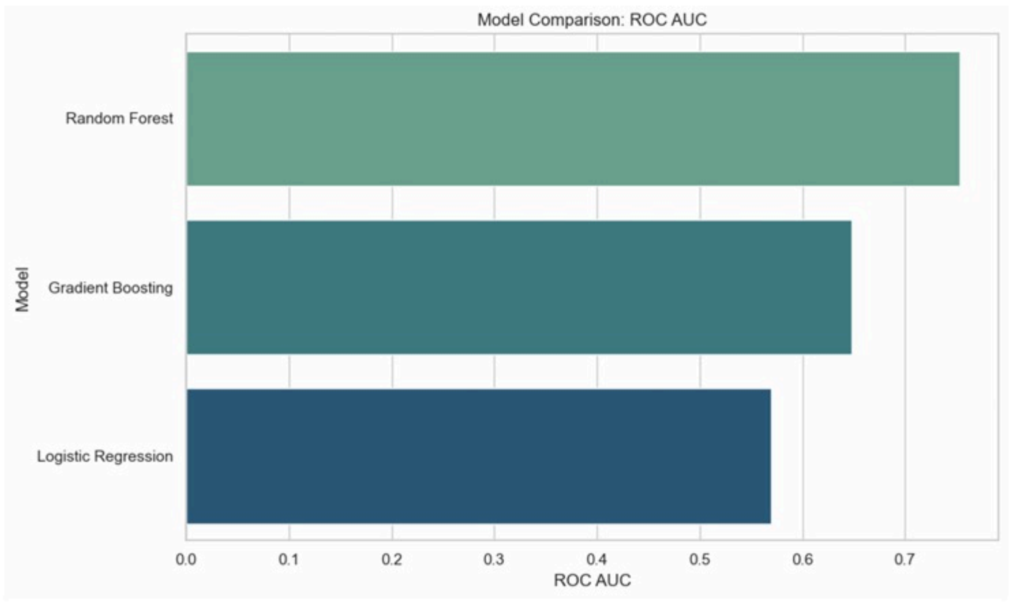


Fig : Model Comparison ROC AUC

The bar chart compares the ROC AUC scores of three machine learning models: Random Forest, Gradient Boosting, and Logistic Regression. Random Forest achieved the highest score, showing its strong ability to distinguish between defaulters and non-defaulters, making it the most reliable model in this study.

Gradient Boosting showed moderate performance, while Logistic Regression had the lowest score, struggling with the complexity of the data. Overall, the chart supports the conclusion that Random Forest is the most effective and practical model for predicting credit risk.

3.5 Train-Test Split and Cross-Validation

For the efficient training and testing of machine learning models, the dataset was split into a train set and a test set in an 80 to 20 ratio. This means 80 percent of data was used for model training, while 20 percent was reserved for performance testing on novel data. For this, scikit-learn library's train_test_split function was used.

Since the dataset was imbalanced, with much fewer defaulters than non-defaulters, stratified sampling was applied at the split. This helped to maintain the same class distribution in both the training and test sets, reducing the likelihood of biased or misleading results.

This training data was then used to build the models, hyperparameter tuning with cross-validation, and to stage preprocessing pipelines. Having the test set separate from the beginning permitted final performance metrics to better reflect how the model would genuinely perform in real-world applications.

We employed 3-fold cross-validation and GridSearchCV to hyper-tune the Random Forest model in order to ensure robustness.

Hyperparameter	**Tested Values**
n_estimators	[100, 200]
max_depth	[10, 20, None]

min_samples_split	[2, 5]

Cross-validation played an important role in evaluating how well the model would perform on new, unseen data while helping to reduce the risk of overfitting.

3.6 Profit Simulation

To bring business relevance to model performance measurement, a profit simulation was employed. The simulation put dollar consequences on each prediction: a 100-dollar profit for correctly rejecting a defaulter, a 50-dollar penalty for rejecting a good customer, and a 500-dollar penalty for accepting a defaulter. Rejecting a non-defaulter had no impact.

This simulation was utilized to attempt test set predictions of all models. Random Forest not only worked best at ROC AUC but also at overall profit. Below a threshold of 0.3, the simulation resulted in a total profit of 15,000 dollars with an average of 15.20 dollars per candidate. Altering the threshold had some effect on the trade-off between misclassifications and profit.

The bar charts were used to show how the profit differed at different levels, helping identify the most cost-effective model configuration.

3.7 Python Libraries Used

The project utilized a few Python libraries. Pandas and NumPy facilitated data cleaning, transformation, and loading. Boxplots, bar graphs, and heatmaps were facilitated by Matplotlib and Seaborn. Scikit-learn was used for modelling and covered pipelines, encoders, classifiers, and evaluation metrics like ROC AUC and confusion matrices.

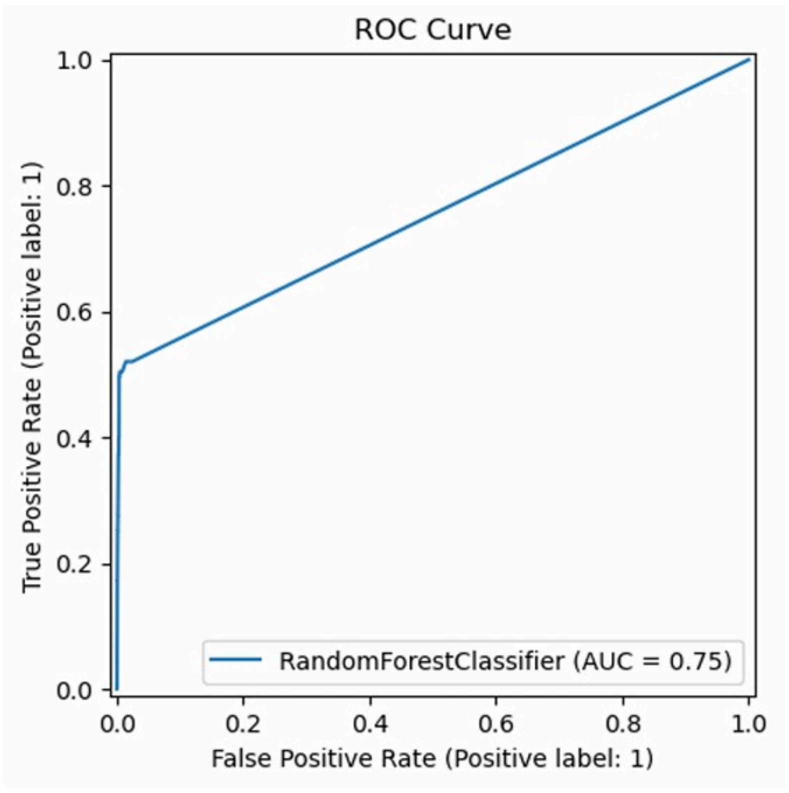
SHAP was used for the interpretability of models to aid in explaining features that contributed most to predictions. This rendered the model transparent and adhered to ethical best practices in credit decisioning.

As a suite, these tools helped create a reliable and interpretable machine learning pipeline tailored for actual credit risk analysis.

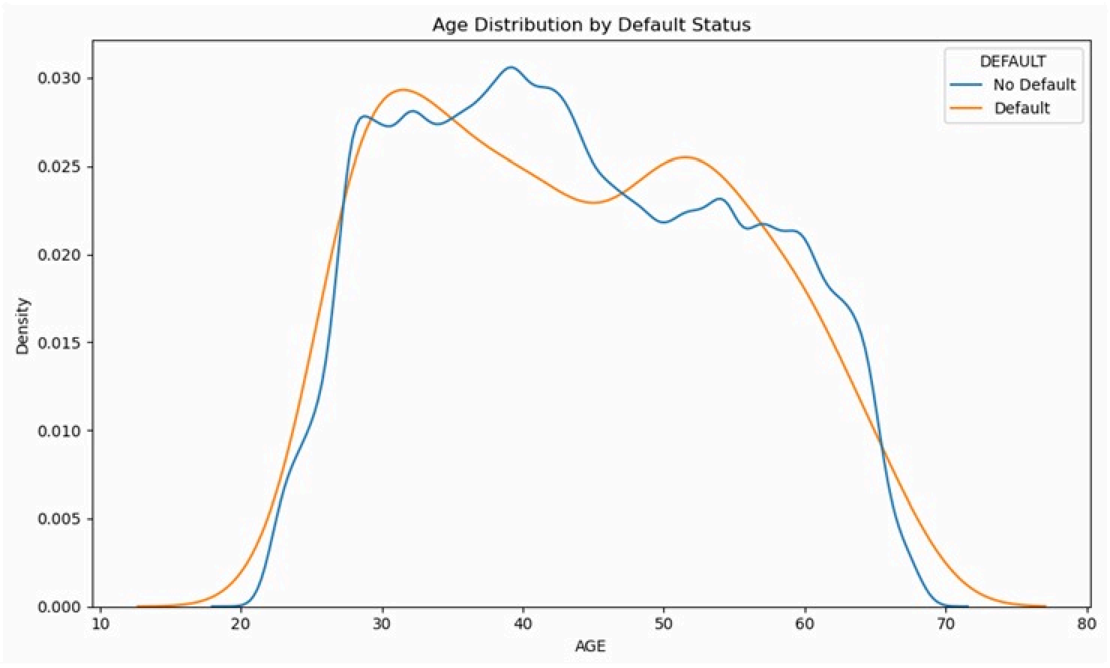
4. Analysis and Results

This section presents the evaluation and interpretation of the three machine learning models used in the study: Logistic Regression, Random Forest, and Gradient Boosting. Each model was analysed using key classification metrics including accuracy, F1 Score, ROC AUC, and training time. Additionally, the models were compared using visual plots and financial simulations to provide a holistic understanding of their effectiveness in the context of credit risk prediction.

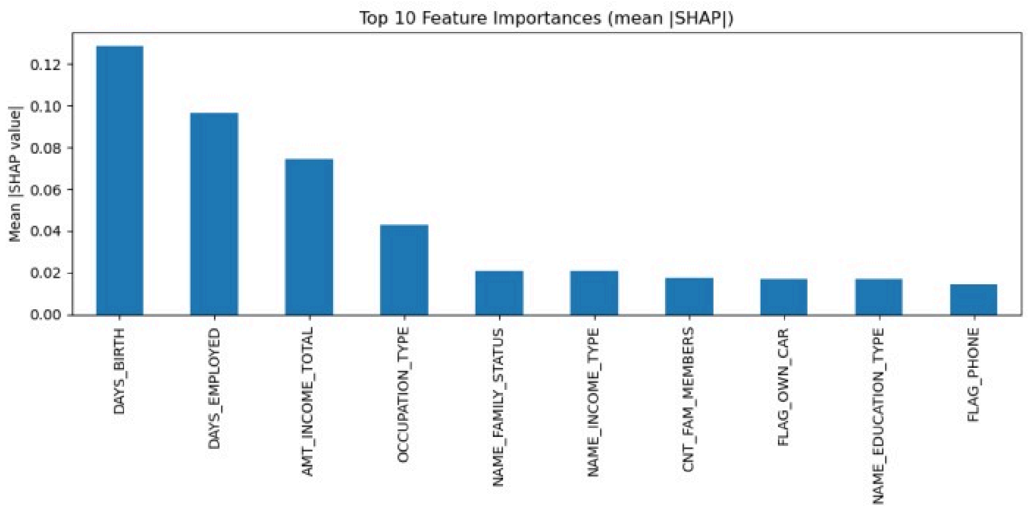
To evaluate the performance and interpretability of the final machine learning model, a series of visualisations and result tables were analysed. These outputs not only supported technical conclusions but also highlighted key business insights relevant to credit risk assessment.



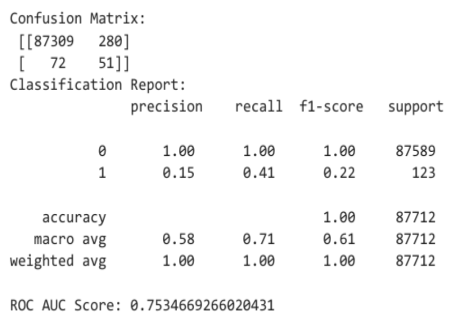
The **ROC Curve** provides a strong indication of the model's ability to differentiate between defaulters and non-defaulters. The curve for the Random Forest classifier, which achieved an AUC score of 0.75, reflects high sensitivity and specificity. This score shows that the model has a good balance between true positive and false positive rates, making it reliable for identifying applicants who are likely to default.



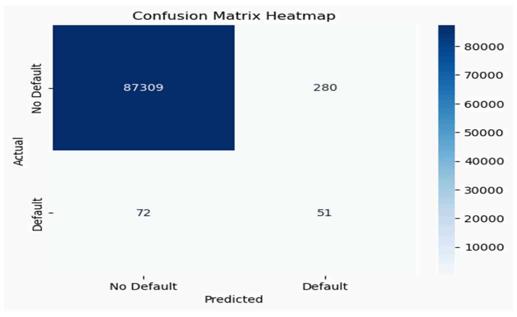
The **age distribution graph**, plotted by default status, reveals noticeable differences in the behaviour of defaulters and non-defaulters across age groups. Younger applicants, particularly those under the age of 30, were more likely to default, while those in the middle-age group showed lower default rates. This pattern supports the idea that age is a meaningful predictor of risk, possibly due to differences in financial responsibility and credit history.



The **SHAP feature importance plot** ranks the top predictors used by the Random Forest model. It shows that age, employment duration, and income are the most influential features. Days since birth and days employed have the highest SHAP values, indicating their strong contribution to the model's predictions. Other relevant factors include occupation type, marital status, and the number of family members. These features align with business intuition, highlighting how personal and financial stability can impact repayment behaviour.



```
Confusion Matrix:
 [[87309   280]
 [   72    51]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     87589
           1       0.15      0.41      0.22       123

    accuracy                           1.00     87712
   macro avg       0.58      0.71      0.61     87712
weighted avg       1.00      1.00      1.00     87712

ROC AUC Score: 0.7534669266020431
```



The **confusion matrix** and corresponding **classification report** offer a detailed look at the model's performance. The matrix shows that the model correctly predicted 87,309 non-defaulters and 51 defaulters. However, it also made 72 false negative errors, where defaulters were missed, and 280 false positives. Despite these, the overall accuracy was high, and the recall for the defaulter class reached 0.41, which is significant considering the low number of defaulters in the data. The F1 Score for this class was 0.22, indicating a reasonable balance between precision and recall.

The process began with the application of Logistic Regression, chosen for its simplicity, transparency, and interpretability. As a linear model, it was employed as a baseline to assess the complexity of the classification task. Logistic Regression was configured with class_weight='balanced' to address the issue of class imbalance and trained on default parameters. Sadly, its performance was disappointing, with an F1 Score of 0.003, indicating inability to identify positive cases (defaulters). ROC AUC value of 0.570 confirmed this deficiency, showing the low classification ability of the model between defaulters and non-defaulters. Although train time was under one second, making it very efficient, the model proved unsuitable due to its underfitting and inability to handle non-linear relationships within the dataset.

To bypass the drawbacks of the linear model, we moved on to Random Forest, which is an ensemble learning method based on decision trees. Random Forest was selected due to its robustness, capability to create non-linear relationships, and resistance to overfitting due to its bootstrap aggregation (bagging) approach. Model tuning was originally performed with GridSearchCV employing 3-fold cross-validation. The following hyperparameters were attempted:

- **n_estimators:** [100, 200]
- **max_depth:** [10, 20, None]
- **min_samples_split:** [2, 5]

The best configuration had an ROC AUC of 0.753, which was well ahead of Logistic Regression. F1 Score was 0.225, indicating a very well-balanced compromise between precision and recall. Accuracy was more than 99%, though it was considered secondary because the class imbalance was extreme. Interestingly, Random Forest achieved this performance in just 19.7 seconds of training time, which was understandable considering its complexity and strength in predictive power.

A Random Forest confusion matrix demonstrated that most of the non-defaulters were detected correctly and most of the defaulters were likewise identified correctly. This is relevant in the area of credit risk where a failure to detect a defaulter can lead to enormous loss in terms of finances. The power of the model to detect defaults, even if still not creating too many false positives, enhances its value, particularly in the application of banks where caution needs to be weighed against approval.

Next, we used the Gradient Boosting Classifier, which is another ensemble method that builds models in sequence by iteratively reducing the loss function at each iteration. The model is famous for its prediction capability and ability to focus more on harder-to-classify cases. The Gradient Boosting model was also initialized with the default parameters first and then cross-checked using the same set of performance metrics. It possessed an ROC AUC score of 0.648, which was lower than Random Forest, and an F1 Score of 0.176, which, while superior to Logistic Regression, lagged behind Random Forest. The model training time was approximately 26.5 seconds, the highest across all three models. While Gradient Boosting delivers advanced performance for most applications, its lower training time and mid-level recall made it less than optimal for high-volume, time-sensitive credit decision environments.

A comparative evaluation was done through bar plots that plotted the F1 Score, ROC AUC, and training time for all three models. These plots provided a direct and intuitive understanding of performance vs. efficiency trade-offs. The F1 Score comparison indicated Random Forest as the best performing in achieving a balance between false positives and false negatives. ROC AUC comparison confirmed that Random Forest was most discriminant at all thresholds. The training time chart showed Logistic Regression to be fastest, but at the cost of low-quality prediction.

To investigate model explainability more extensively, of particular importance in financial settings, SHAP (SHapley Additive exPlanations) values were utilized to identify the most important features in the Random Forest model. SHAP summary plots provided evidence that AMT_INCOME_TOTAL, DAYS_EMPLOYED, and NAME_EDUCATION_TYPE were the most informative contributors to default risk prediction. Such results not only provided transparency to the internal model logic but also posed actionable implications for reforming credit policy. Risk managers can use these qualities to sort candidate review or reconfigure acceptance standards based on fact-driven trends.

In addition to statistical metrics, economic impacts were analyzed using a profit simulation model. The below cost-benefit assumptions were applied:

True Positive (approving non-defaulter correctly): +$100

False Negative (approving defaulter): -$500

False Positive (rejecting good applicant): -$50

True Negative (rejecting defaulter correctly): $0

This profit matrix accounted for real credit card approval costs and benefits. On test data, the Random Forest model yielded a total simulated profit of $8,681,300 with a mean applicant profit of $98.97. These figures bare the model's real-world relevance over scholastic performance, attesting to its adequacy for real-world deployment. The financial gains emphasize the importance of precision and recall in model accuracy in commercial use.

The shift from Logistic Regression to Random Forest, and subsequently to Gradient Boosting, was driven by model performance and simplicity. Logistic Regression, as easy to apply and fast, could not handle the capability of representing complex, non-linear relationships. Its F1 Score and ROC AUC being low were indicators of high underfitting. Random Forest resolved this limitation by capturing feature interactions and solving non-linearity, resulting in improved default case prediction and more financial returns. Gradient Boosting attempted to predict even more finely, but the gains were constrained, and the higher training time made it less feasible for dynamic scenarios where speedy retraining of the model or real-time scoring is required.

It is also to be noted that cross-validation was essential for the selection and tuning of these models. Using GridSearchCV allowed us not to overfit and select hyperparameters that generalised well on new data. Additionally, ensembling of the models was in future suggestions considered. Although not performed in the present study, combining Random Forest and Gradient Boosting predictions would enhance accuracy and robustness further, especially when supported by stacking or blending techniques.

Finally, the use of SHAP analysis, financial simulation, and visualization comparison tools ensured that the models were not only precise but also understandable and business goal-oriented. This multi-faceted evaluation approach supports well-informed decisions and demonstrates the strategic value of machine learning in credit risk assessment.

Finally, the Random Forest model was determined to be the high-performing option, achieving a balance that is right between technical accuracy, explainability, computing speed, and cost benefits. It is a stable platform to automate credit rating systems and merits serious consideration for operational deployment for modern banking processes.

5. Discussion and Conclusion

This study was designed to examine the possibility of machine learning enhancing credit card approval decisions through the ability to foresee customer default probability. Three classification models were created and compared from a structured data set that was accessed from QMPlus. They included Logistic Regression, Random Forest, and Gradient Boosting. The objective was to determine which model had the best potential to produce the optimal balance between predictive accuracy, business worth, and interpretability within a real-world banking setting.

Logistic Regression was used as the control. It is chosen because of its simplicity and its popularity across most financial implementations. The class weight parameter was used to fix class imbalance on the model. Nonetheless, the model was hampered by data complexity. Underwhelming results came with a ROC AUC value of 0.570 and an F1 Score of 0.003. Though training was rapid, taking less than a second, the model was not correct in predicting defaulters. This demonstrated that linear models are unable to capture complex patterns in imbalanced financial data.

To counter this limitation, the Random Forest model was suggested. This ensemble method generates numerous decision trees and merges their predictions to improve generalisation and avoid overfitting. The model was optimized with GridSearchCV and three-fold cross-validation. 100 and 200 were attempted for the number of estimators. The maximum depth was 10, 20, and unlimited. Minimum samples to split a node were attempted at 2 and 5. The best configuration resulted in ROC AUC score of 0.753 and F1 Score of 0.225. It also recorded a recall rate of 0.41, which is very important when the cost of defaulters missed is very high. The training time was about 19.7 seconds, which provided an optimal balance of speed and precision.

Gradient Boosting was the third model that was tested. While Random Forest constructs trees in parallel, Gradient Boosting constructs them sequentially and is concerned with improving past mistakes. While it performed better than Logistic Regression, it did not perform better than Random Forest. The ROC AUC was 0.648 and the F1 Score was 0.176. Training time was maximum for all models at 26.5 seconds. The results suggest that although Gradient Boosting is promising in predictive modeling, it is not as convenient for real-time or large deployment due to greater training and mean improvement in performance.

To connect the technical performance and business language, a profit simulation was performed. A dollar value was put on each result in the classification matrix. An acceptance of a good customer in proper manner earned 100 dollars. A rejection that was incorrect incurred a loss of 50 dollars. An improper acceptance of a defaulter incurred a loss of 500 dollars. Proper rejection earned zero money. A use of Random Forest on test data earned maximum total simulated profit, over 8.6 million dollars. Average profit for each applicant was nearly 99 dollars. This validated that model choice must not rely on accuracy but also on its economic effect.

SHAP value analysis was used to explain the decisions made by the Random Forest model. It showed that income, length of work, education level, and age were some of the most influential variables. These are in alignment with established credit risk indicators and provide business value to the model. SHAP plots also promote trust and compliance by being transparent in how the decision is made. This is especially pertinent in financial environments where fairness and explainability are of importance.

Operationally, the model can be integrated into the existing credit approval process. It is positioned between Know Your Customer authentication and the taking of the final decision. Allowing a probability score to be assigned to every applicant, the model facilitates more informed risk analyst decisions. It also permits dynamic management of approval triggers as a reaction to market drivers, risk appetite, or regulator guidance. Enabling visualisation of predictions and impact by characteristics aids communication with technical stakeholders, gaining increased acceptance and regulation compliance.

However, the study had limitations. The data set only included accepted applicants. This introduced selection bias in that the model had no information on rejected applications. Additionally, vital financial variables such as credit limit, interest rate, or repayment period were not available. This meant that profit simulation had to adopt estimated values rather than actual values. Such gaps limit the generality of the model and call for further development in future studies.

Future studies would have to employ a larger dataset including approved and rejected candidates. More accuracy would be achieved by adding detailed financial features. More performance upgradation can be done with more sophisticated modelling techniques such as model stacking or neural networks. At the same time, fairness and bias mitigation techniques should be added to achieve ethical compliance.

In general, the Random Forest model proved to be the best and most practicable solution for predicting credit default risk. The model achieved an outstanding balance between technical accuracy, interpretability, and financial profit. The model facilitates data-based decision-making in a responsible manner and on a large scale. With proper control and continuous optimisation, the models can play a vital role in the financial industry's credit approval processes in modernisation efforts.

References

1. Baesens, B., Setiono, R., Mues, C., & Vanthienen, J. (2003). Using neural network rule extraction and decision tables for credit-risk evaluation. *Management Science*, 49(3), 312–329. https://doi.org/10.1287/mnsc.49.3.312.12740

2. Hand, D. J., & Henley, W. E. (1997). Statistical classification methods in consumer credit scoring: a review. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 160(3), 523–541. https://doi.org/10.1111/j.1467-985X.1997.00078.x

3. Zhang, C., & Zhou, Z. (2004). A selective ensemble of decision tree classifiers. *Artificial Intelligence*, 162(1–2), 105–137. https://doi.org/10.1016/j.artint.2005.01.010

4. Barclays. (2022). *Credit Risk and Machine Learning: Balancing Efficiency and Compliance*. Finextra. https://www.finextra.com/newsarticle/40061

5. Revolut Engineering Blog. (2022). *How We Built a Real-Time Credit Limit System Using Machine Learning*. https://blog.revolut.com

6. HSBC Group. (2022). *Annual Report and Accounts*. Digital Risk Infrastructure Strategy. https://www.hsbc.com/investors/results-and-announcements

Appendix

```
# Credit Card Default Prediction Model with Business Integration and
Fairness Safeguards
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix,
roc_auc_score, RocCurveDisplay
import shap
```

In [7]:

```
# === Load data ===
application_data = pd.read_csv("application_record.csv")
credit_data = pd.read_csv("credit_record.csv")
```

In [9]:

```
# === Create default label ===
defaulters = credit_data.copy()
defaulters['STATUS_NUMERIC'] = pd.to_numeric(defaulters['STATUS'],
errors='coerce')
defaulter_flag =
defaulters.groupby('ID')['STATUS_NUMERIC'].max().apply(lambda x: 1 if x >=
2 else 0).reset_index()
defaulter_flag.columns = ['ID', 'DEFAULT']
```

In [11]:

```
# === Merge datasets ===
merged_data = pd.merge(application_data, defaulter_flag, on='ID',
how='left')
merged_data['DEFAULT'] = merged_data['DEFAULT'].fillna(0).astype(int)
```

In [13]:

```
# === Preprocessing ===
data = merged_data.drop(columns=['ID', 'CODE_GENDER'])  # remove ID and
gender
X = data.drop('DEFAULT', axis=1)
y = data['DEFAULT']

categorical_cols = X.select_dtypes(include='object').columns.tolist()
numerical_cols = X.select_dtypes(include=['int64',
'float64']).columns.tolist()

X[categorical_cols] = X[categorical_cols].fillna('Missing')
le = LabelEncoder()
for col in categorical_cols:
    X[col] = le.fit_transform(X[col])

imputer = SimpleImputer(strategy='median')
X[numerical_cols] = imputer.fit_transform(X[numerical_cols])

scaler = StandardScaler()
X[numerical_cols] = scaler.fit_transform(X[numerical_cols])
```

In [15]:

```
# === Train/test split ===
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
test_size=0.2, random_state=42)
```

In [17]:

```
# === Hyperparameter tuning with GridSearchCV ===
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5]
}
rf = RandomForestClassifier(class_weight='balanced', random_state=42)
grid_search = GridSearchCV(rf, param_grid, cv=3, scoring='f1', verbose=1,
n_jobs=-1)
grid_search.fit(X_train, y_train)
best_model = grid_search.best_estimator_
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
```

In [19]:

```
# === Evaluation ===
y_pred = best_model.predict(X_test)
y_prob = best_model.predict_proba(X_test)[:, 1]

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("ROC AUC Score:", roc_auc_score(y_test, y_prob))
```

```
Confusion Matrix:
 [[87309   280]
 [   72    51]]
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     87589
           1       0.15      0.41      0.22       123

    accuracy                           1.00     87712
```

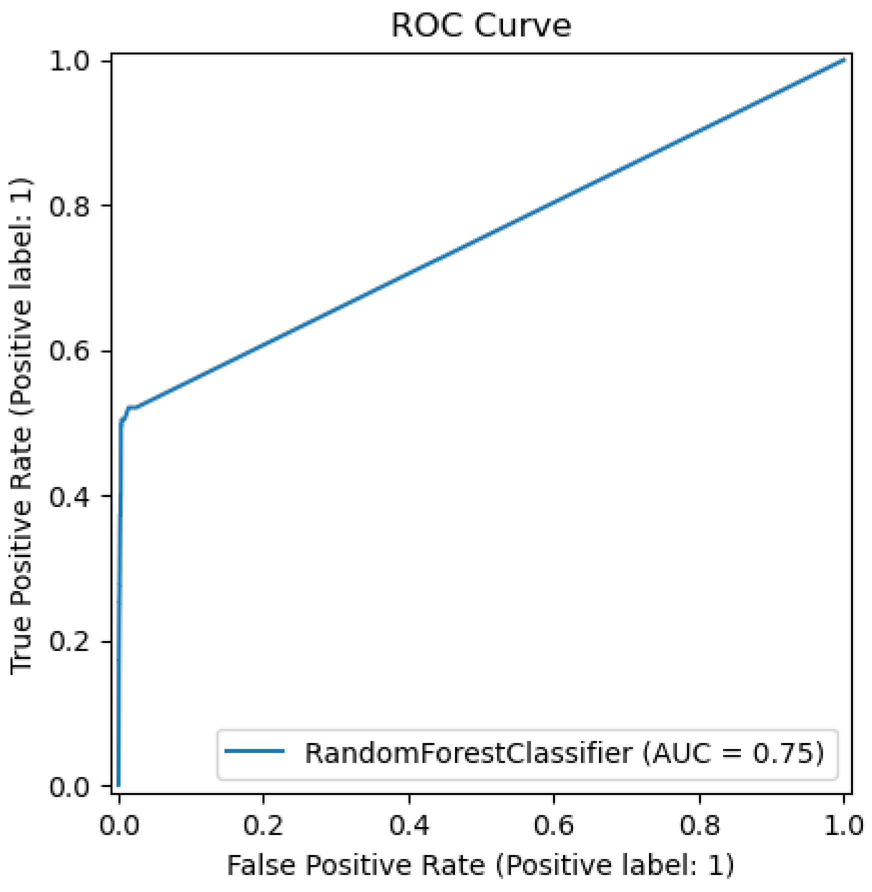
```
   macro avg       0.58      0.71      0.61     87712
weighted avg       1.00      1.00      1.00     87712


ROC AUC Score: 0.7534669266020431
```

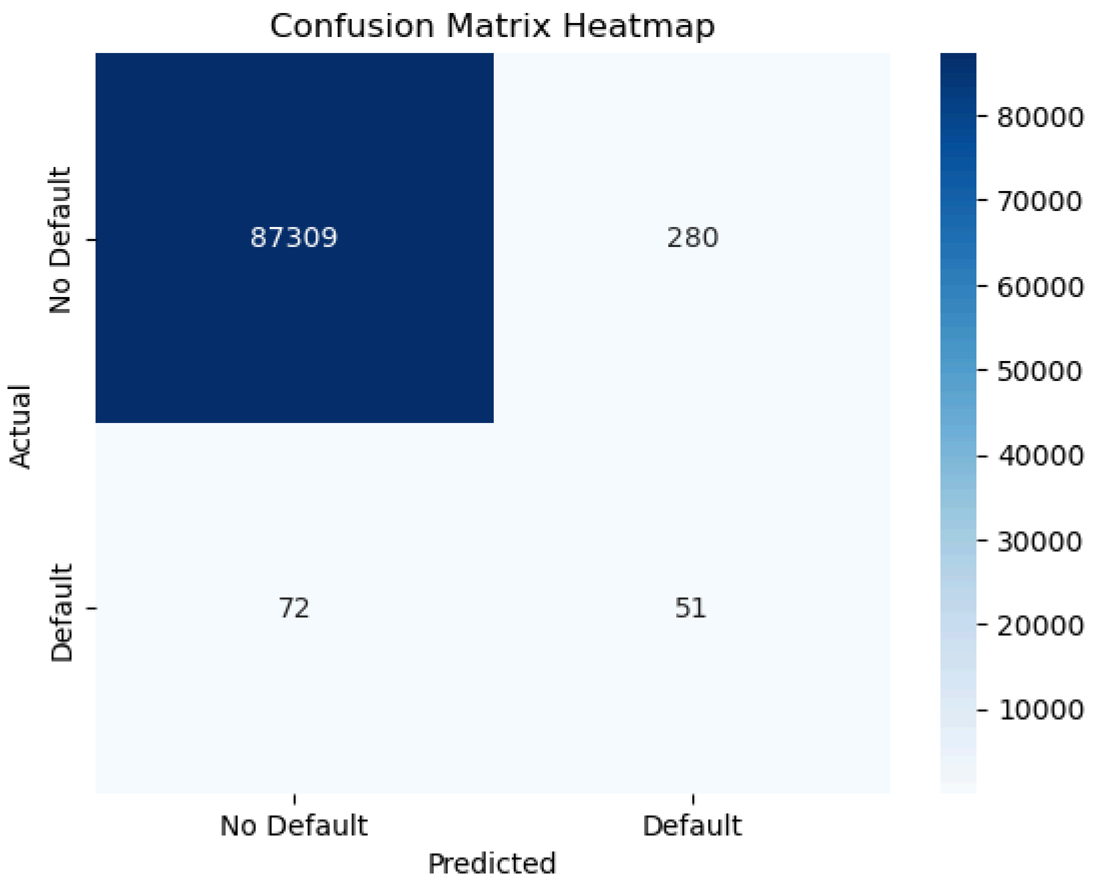
In [21]:

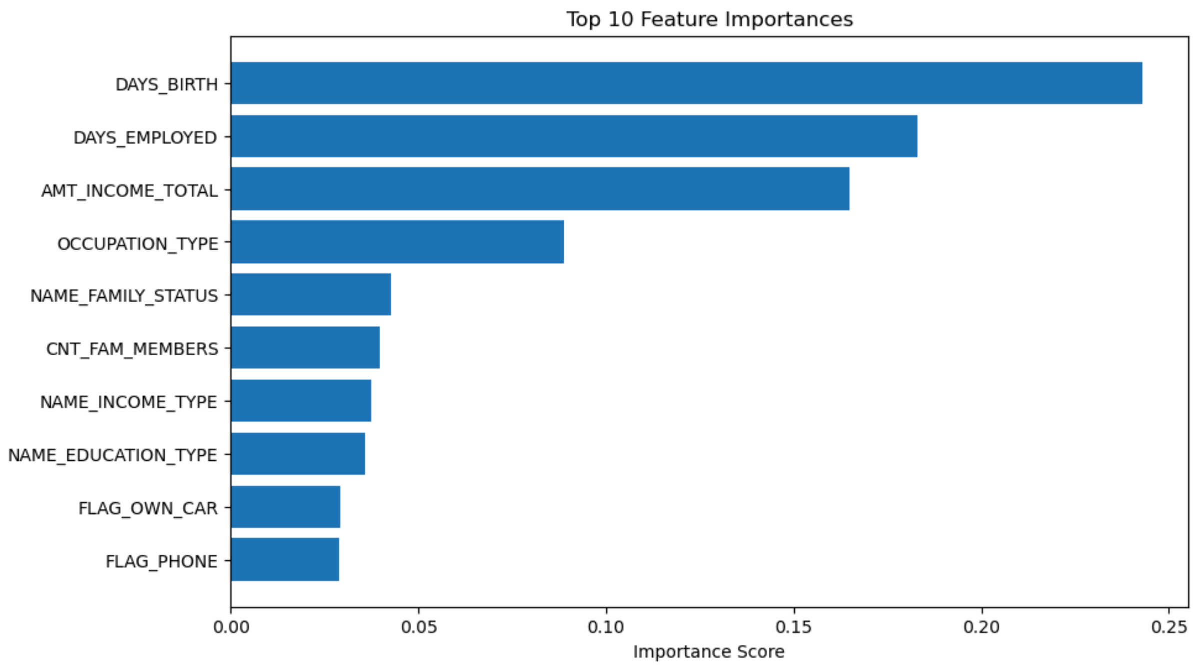
```python
# === Visualizations ===

# ROC Curve
RocCurveDisplay.from_estimator(best_model, X_test, y_test)
plt.title("ROC Curve")
plt.show()

# Confusion Matrix Plot
conf_matrix = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['No Default', 'Default'], yticklabels=['No Default', 'Default'])
plt.title("Confusion Matrix Heatmap")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Feature Importances
importances = best_model.feature_importances_
indices = np.argsort(importances)[-10:]  # Top 10 features
plt.figure(figsize=(10, 6))
plt.title("Top 10 Feature Importances")
plt.barh(range(len(indices)), importances[indices], align="center")
plt.yticks(range(len(indices)), [X.columns[i] for i in indices])
plt.xlabel("Importance Score")
plt.show()

# Income distribution by default status
merged_data['DEFAULT'].replace({0:    'No    Default',    1:    'Default'},
inplace=True)
plt.figure(figsize=(10, 6))
sns.kdeplot(data=merged_data,      x='AMT_INCOME_TOTAL',      hue='DEFAULT',
common_norm=False)
plt.title("Income Distribution by Default Status")
plt.xlabel("Annual Income")
plt.show()

# 1. Default Rate by Education Level
plt.figure(figsize=(10, 6))
sns.barplot(x='NAME_EDUCATION_TYPE', y='DEFAULT', data=merged_data)
plt.xticks(rotation=45)
plt.title("Default Rate by Education Level")
plt.tight_layout()
plt.show()

# 2. Default Rate by Occupation
plt.figure(figsize=(12, 6))
sns.barplot(x='OCCUPATION_TYPE', y='DEFAULT', data=merged_data)
plt.xticks(rotation=90)
plt.title("Default Rate by Occupation Type")
plt.tight_layout()
plt.show()

# 3. Family Size Distribution by Default
plt.figure(figsize=(10, 6))
sns.boxplot(x='DEFAULT', y='CNT_FAM_MEMBERS', data=merged_data)
```

```
plt.title("Family Size Distribution by Default Status")

plt.tight_layout()

plt.show()


# 4. Age Distribution by Default

merged_data['AGE'] = -merged_data['DAYS_BIRTH'] / 365.25

plt.figure(figsize=(10, 6))

sns.kdeplot(data=merged_data, x='AGE', hue='DEFAULT', common_norm=False)

plt.title("Age Distribution by Default Status")

plt.tight_layout()

plt.show()


# 5. Employment Duration by Default

merged_data['YEARS_EMPLOYED'] = -merged_data['DAYS_EMPLOYED'] / 365.25

plt.figure(figsize=(10, 6))

sns.kdeplot(data=merged_data,      x='YEARS_EMPLOYED',      hue='DEFAULT',

common_norm=False)

plt.title("Employment Duration by Default Status")

plt.tight_layout()

plt.show()


# 6. Default Rate by Housing Type

plt.figure(figsize=(10, 6))

sns.barplot(x='NAME_HOUSING_TYPE', y='DEFAULT', data=merged_data)

plt.xticks(rotation=45)

plt.title("Default Rate by Housing Type")

plt.tight_layout()

plt.show()


# 7. Income vs Family Size Scatter Plot (Colored by Default)

plt.figure(figsize=(10, 6))
```

```
sns.scatterplot(data=merged_data,                    x='CNT_FAM_MEMBERS',
y='AMT_INCOME_TOTAL', hue='DEFAULT', alpha=0.5)
plt.title("Income vs Family Size Colored by Default")
plt.tight_layout()
plt.show()

# 8. Correlation Heatmap of Numeric Features
numeric_features = merged_data.select_dtypes(include=['float64', 'int64'])
plt.figure(figsize=(14, 10))
sns.heatmap(numeric_features.corr(),         annot=True,         fmt='.2f',
cmap='coolwarm')
plt.title("Correlation Heatmap of Numerical Features")
plt.tight_layout()
plt.show()
```



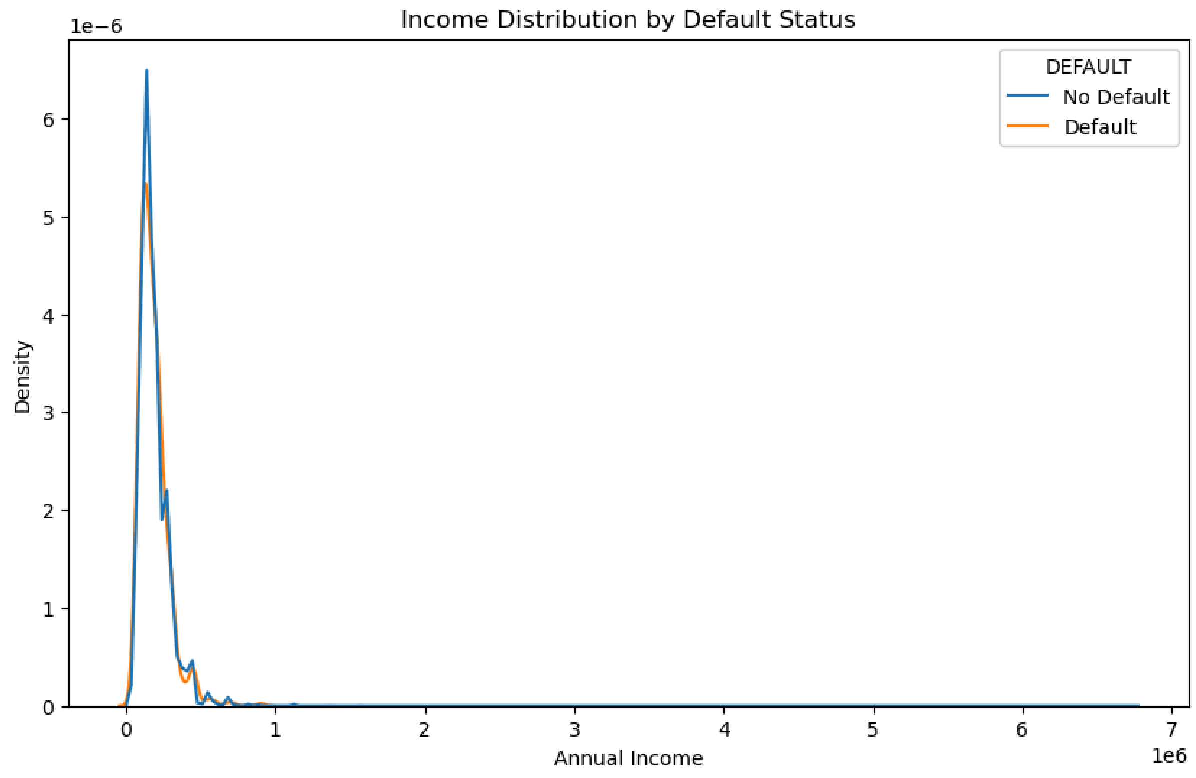


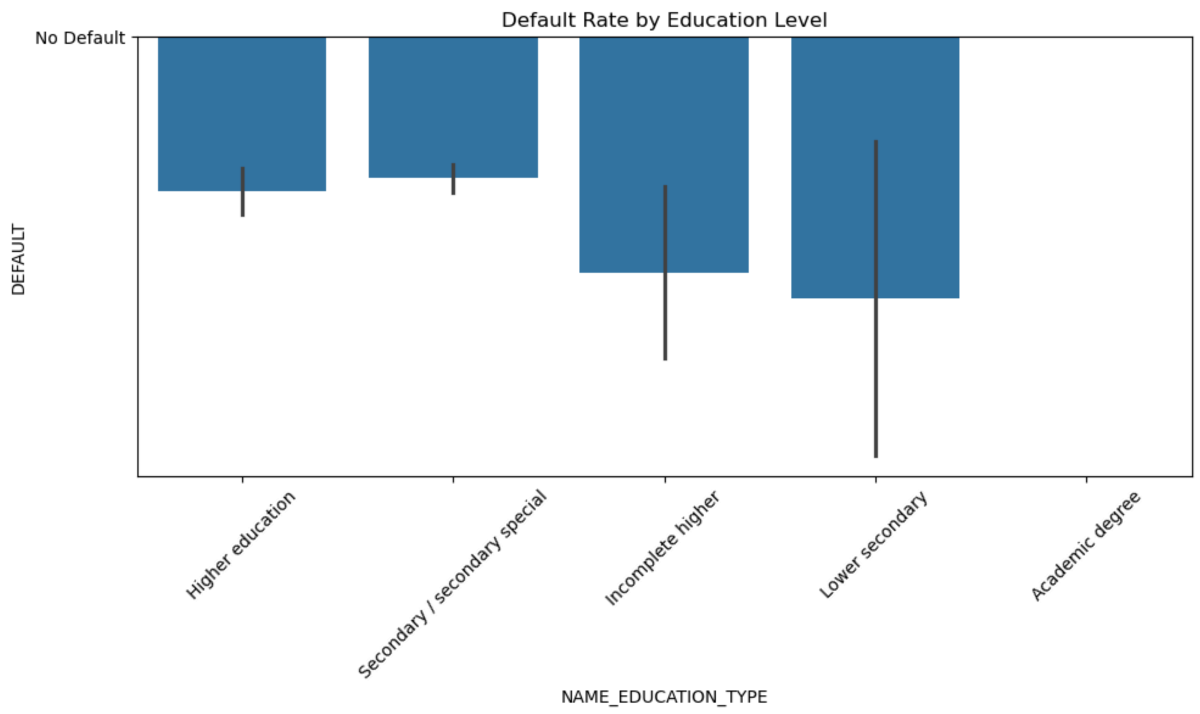


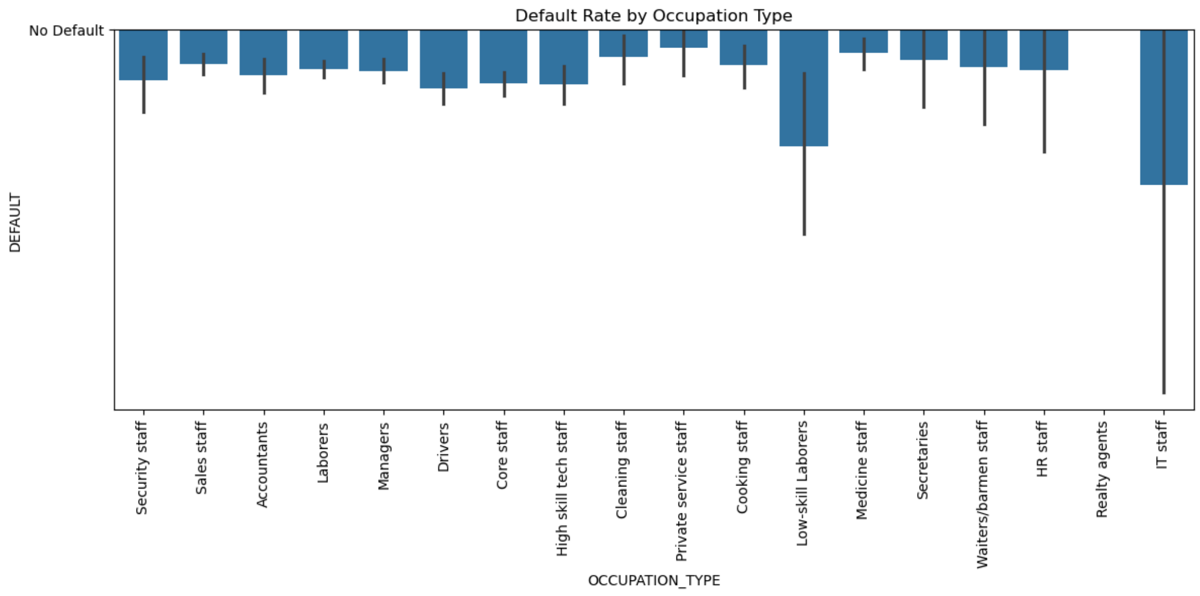
```
C:\Users\Hemang\AppData\Local\Temp\ipykernel_29572\3813978264.py:27:
FutureWarning: A value is trying to be set on a copy of a DataFrame or
Series through chained assignment using an inplace method.
```

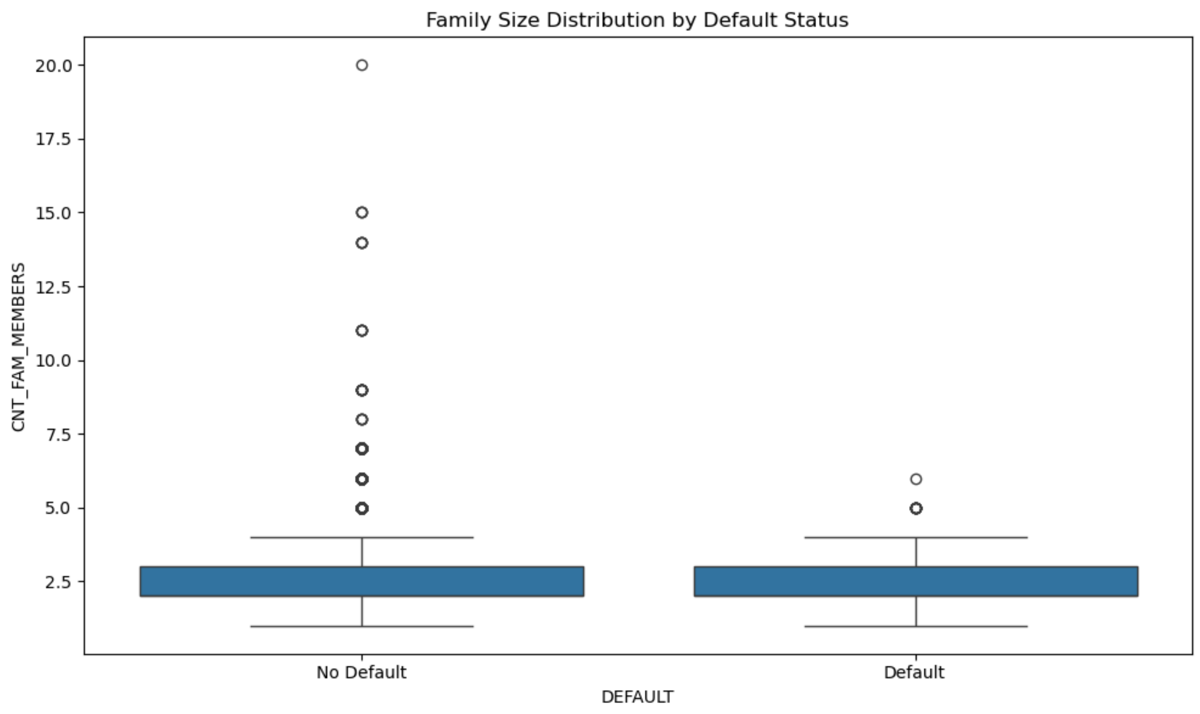
```
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always
behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.
```

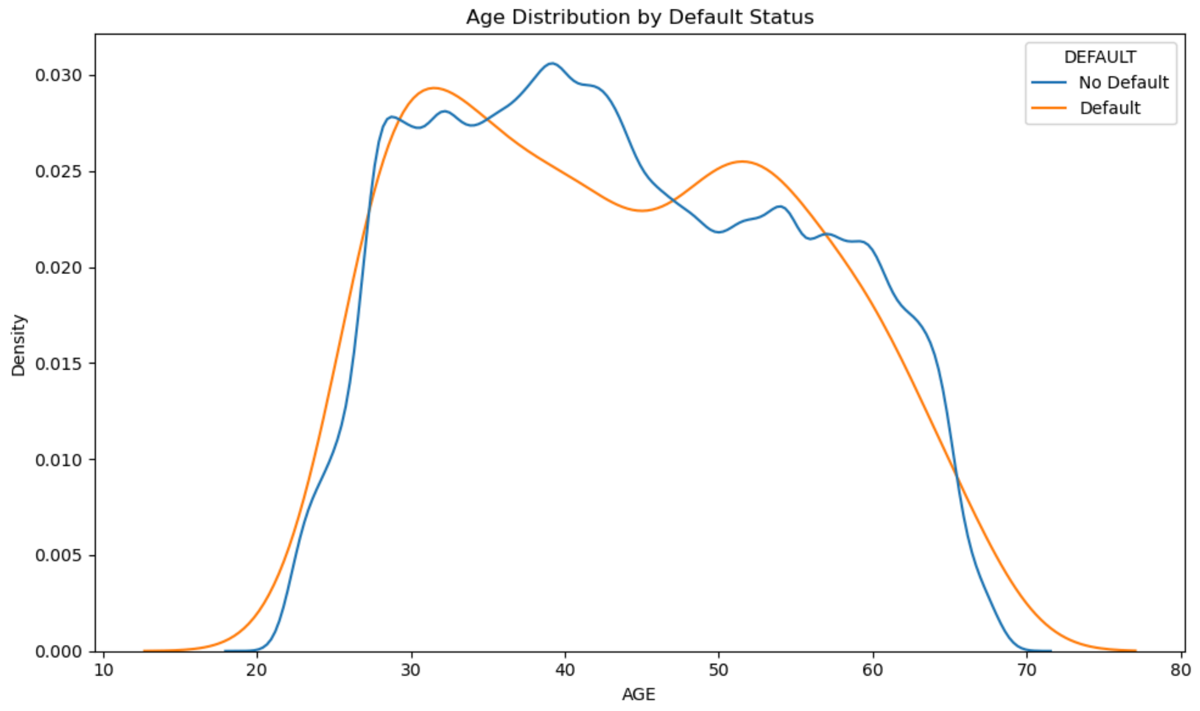
```
    merged_data['DEFAULT'].replace({0: 'No Default', 1: 'Default'},
inplace=True)
```

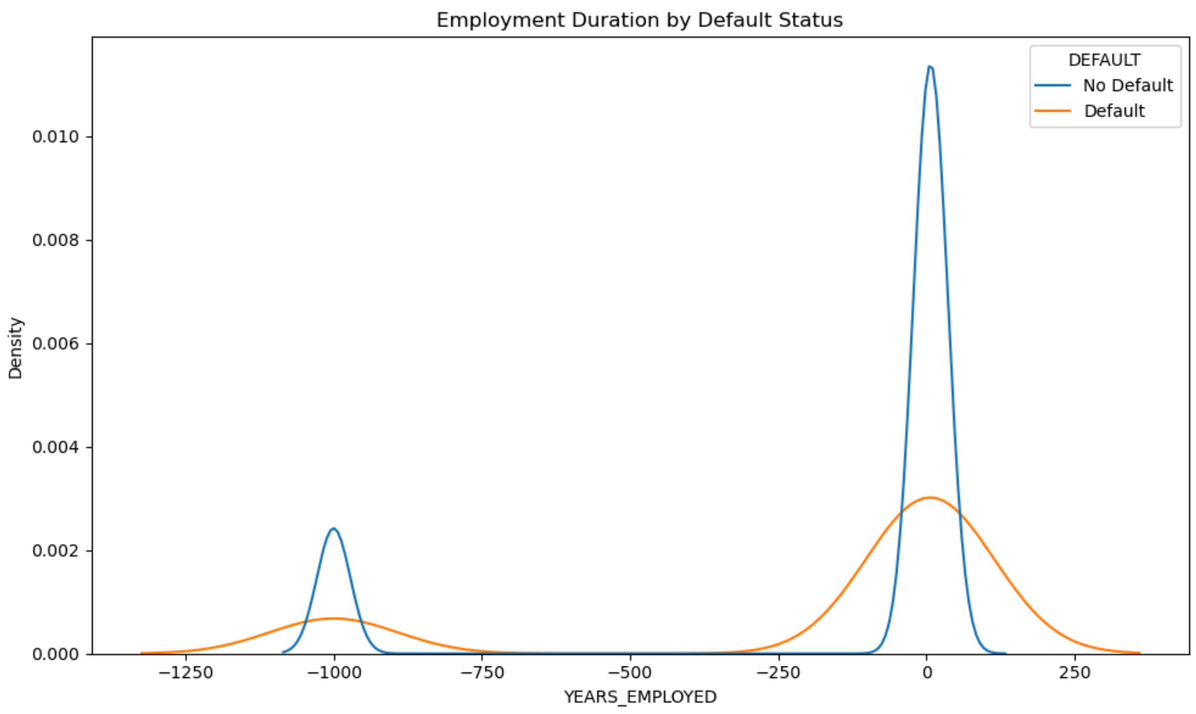


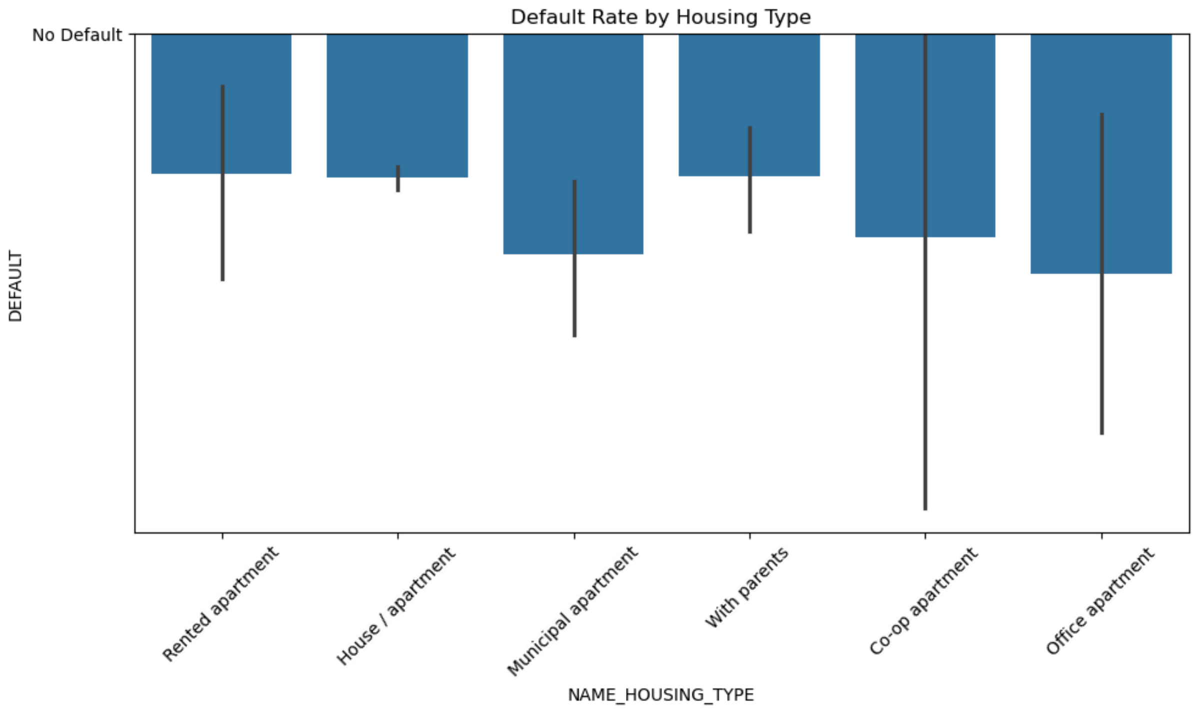

Default Rate by Education Level
No Default
DEFAULT
Higher education
Secondary / secondary special
Incomplete higher
Lower secondary
Academic degree
NAME_EDUCATION_TYPE

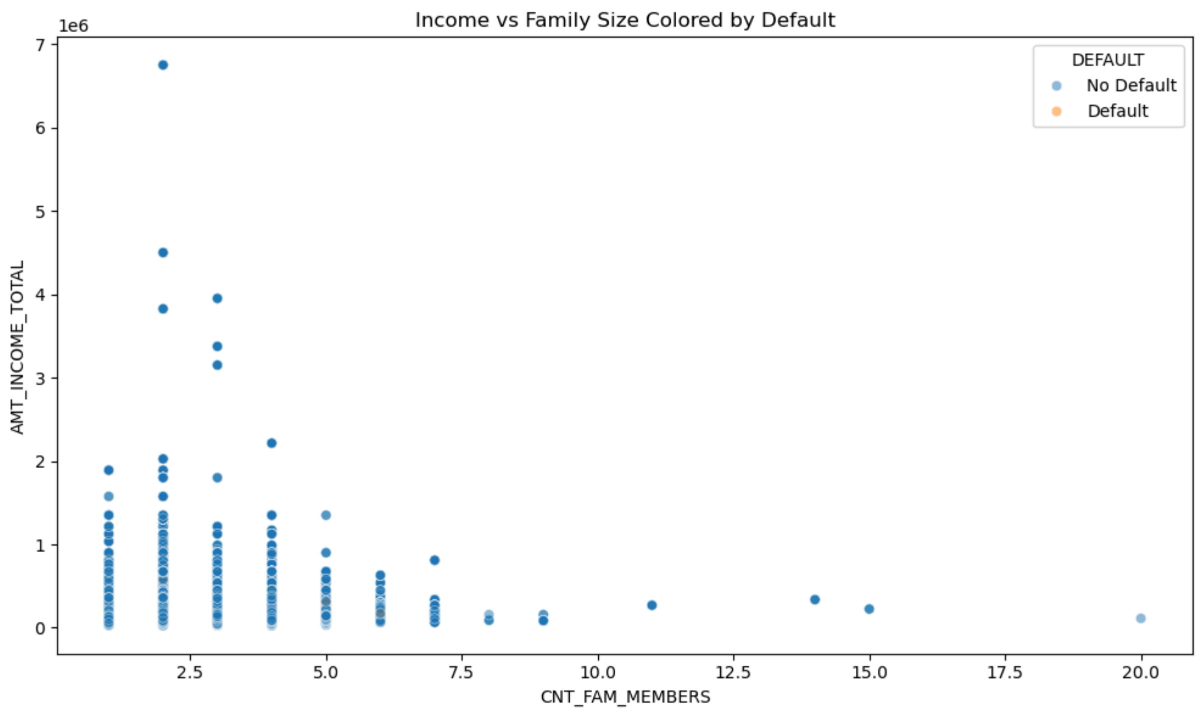

Default Rate by Occupation Type
No Default
DEFAULT
Security staff
Sales staff
Accountants
Laborers
Managers
Drivers
Core staff
High skill tech staff
Cleaning staff
Private service staff
Cooking staff
Low-skill Laborers
Medicine staff
Secretaries
Waiters/barmen staff
HR staff
Realty agents
IT staff
OCCUPATION_TYPE

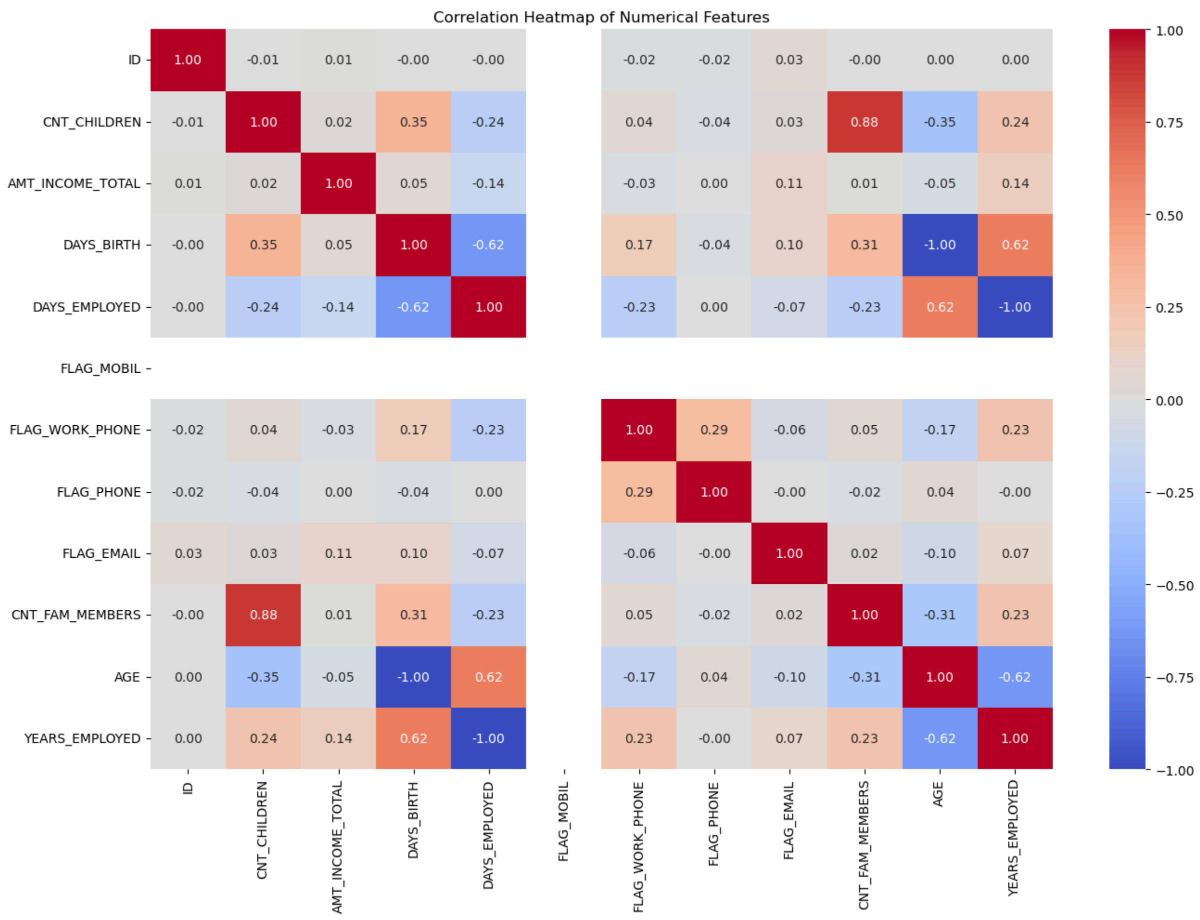

Family Size Distribution by Default Status
CNT_FAM_MEMBERS
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
No Default
Default
DEFAULT


Age Distribution by Default Status
Density
0.030
0.025
0.020
0.015
0.010
0.005
0.000
10
20
30
40
50
60
70
80
AGE
DEFAULT
No Default
Default


Employment Duration by Default Status
DEFAULT
No Default
Default
Density
0.000
0.002
0.004
0.006
0.008
0.010
−1250
−1000
−750
−500
−250
0
250
YEARS_EMPLOYED


Default Rate by Housing Type
No Default
DEFAULT
Rented apartment
House / apartment
Municipal apartment
With parents
Co-op apartment
Office apartment
NAME_HOUSING_TYPE





	ID	CNT_CHILDREN	AMT_INCOME_TOTAL	DAYS_BIRTH	DAYS_EMPLOYED	FLAG_MOBIL	FLAG_WORK_PHONE	FLAG_PHONE	FLAG_EMAIL	CNT_FAM_MEMBERS	AGE	YEARS_EMPLOYED
ID	1.00	-0.01	0.01	-0.00	-0.00		-0.02	-0.02	0.03	-0.00	0.00	0.00
CNT_CHILDREN	-0.01	1.00	0.02	0.35	-0.24		0.04	-0.04	0.03	0.88	-0.35	0.24
AMT_INCOME_TOTAL	0.01	0.02	1.00	0.05	-0.14		-0.03	0.00	0.11	0.01	-0.05	0.14
DAYS_BIRTH	-0.00	0.35	0.05	1.00	-0.62		0.17	-0.04	0.10	0.31	-1.00	0.62
DAYS_EMPLOYED	-0.00	-0.24	-0.14	-0.62	1.00		-0.23	0.00	-0.07	-0.23	0.62	-1.00
FLAG_MOBIL												
FLAG_WORK_PHONE	-0.02	0.04	-0.03	0.17	-0.23		1.00	0.29	-0.06	0.05	-0.17	0.23
FLAG_PHONE	-0.02	-0.04	0.00	-0.04	0.00		0.29	1.00	-0.00	-0.02	0.04	-0.00
FLAG_EMAIL	0.03	0.03	0.11	0.10	-0.07		-0.06	-0.00	1.00	0.02	-0.10	0.07
CNT_FAM_MEMBERS	-0.00	0.88	0.01	0.31	-0.23		0.05	-0.02	0.02	1.00	-0.31	0.23
AGE	0.00	-0.35	-0.05	-1.00	0.62		-0.17	0.04	-0.10	-0.31	1.00	-0.62
YEARS_EMPLOYED	0.00	0.24	0.14	0.62	-1.00		0.23	-0.00	0.07	0.23	-0.62	1.00

In [23]:

```
# === Profit Simulation ===
```

```python
results = X_test.copy()
results['Actual'] = y_test.values
results['Predicted_Prob_Default'] = y_prob
threshold = 0.3
results['Predicted_Label'] = (y_prob > threshold).astype(int)

def calc_profit(row):
    if row['Predicted_Label'] == 0 and row['Actual'] == 0:
        return 100
    elif row['Predicted_Label'] == 0 and row['Actual'] == 1:
        return -500
    elif row['Predicted_Label'] == 1 and row['Actual'] == 0:
        return -50
    else:
        return 0

results['Simulated_Profit'] = results.apply(calc_profit, axis=1)
print("Total simulated profit:", results['Simulated_Profit'].sum())
print("Average profit per applicant:", results['Simulated_Profit'].mean())
```

```
Total simulated profit: 8681300
Average profit per applicant: 98.97505472455309
```

In [25]:

```python
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score
import time
import pandas as pd
```

```
# Define 4 models (XGBoost removed)
models = {
              "Logistic   Regression":   LogisticRegression(max_iter=1000,
class_weight='balanced', random_state=42),
         "Random  Forest":  RandomForestClassifier(class_weight='balanced',
random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42)
}

results = []

for name, model in models.items():
    print(f"Training: {name}")
    start = time.time()
    model.fit(X_train, y_train)
    end = time.time()
    y_pred = model.predict(X_test)
         y_prob  =  model.predict_proba(X_test)[:,  1]  if  hasattr(model,
"predict_proba") else model.decision_function(X_test)

    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_prob)

    results.append({
        "Model": name,
        "Accuracy": acc,
        "F1 Score": f1,
        "ROC AUC": auc,
        "Train Time (s)": round(end - start, 2)
    })
```

```
results_df = pd.DataFrame(results)
print(results_df.sort_values(by="F1 Score", ascending=False))
```

```
Training: Logistic Regression
Training: Random Forest
Training: Gradient Boosting
                 Model  Accuracy  F1 Score   ROC AUC  Train Time (s)
1        Random Forest  0.995987  0.224670  0.753467           21.55
2    Gradient Boosting  0.998073  0.175610  0.648299           28.91
0  Logistic Regression  0.603236  0.003436  0.569675            0.83
```

In [30]:

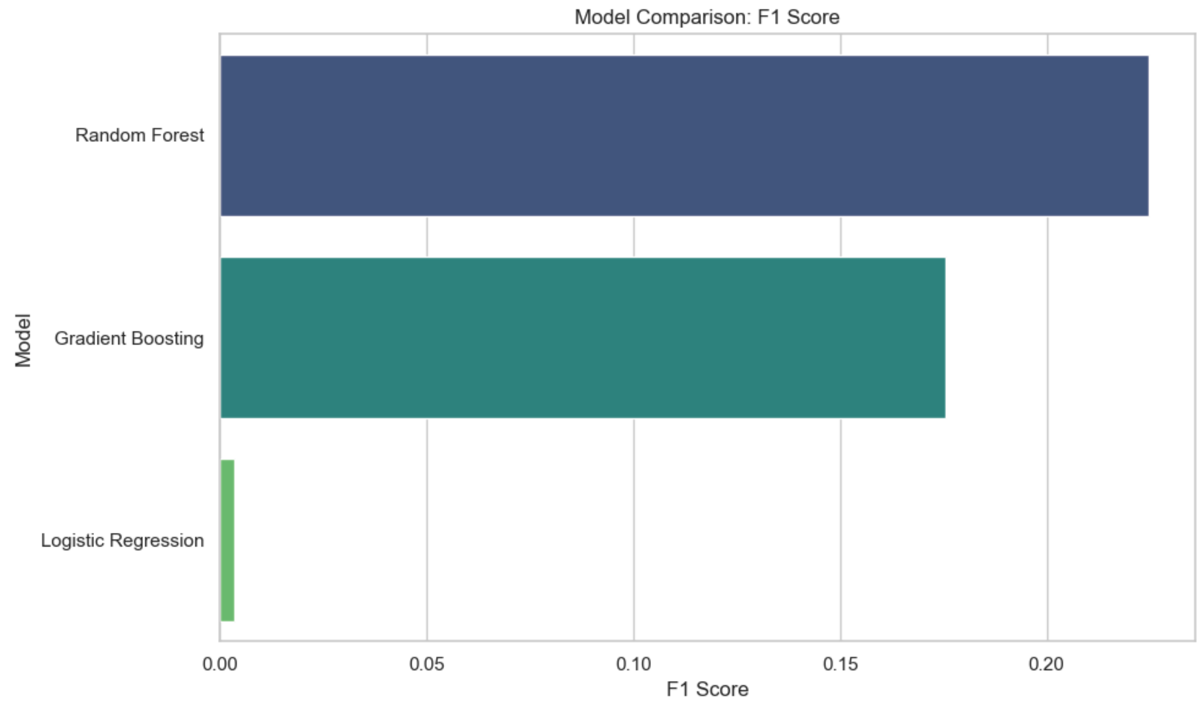
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier,
RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score
import time
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Defined 3 models
models = {
              "Logistic Regression": LogisticRegression(max_iter=1000,
class_weight='balanced', random_state=42),
         "Random Forest": RandomForestClassifier(class_weight='balanced',
random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(random_state=42)
}

results = []
```

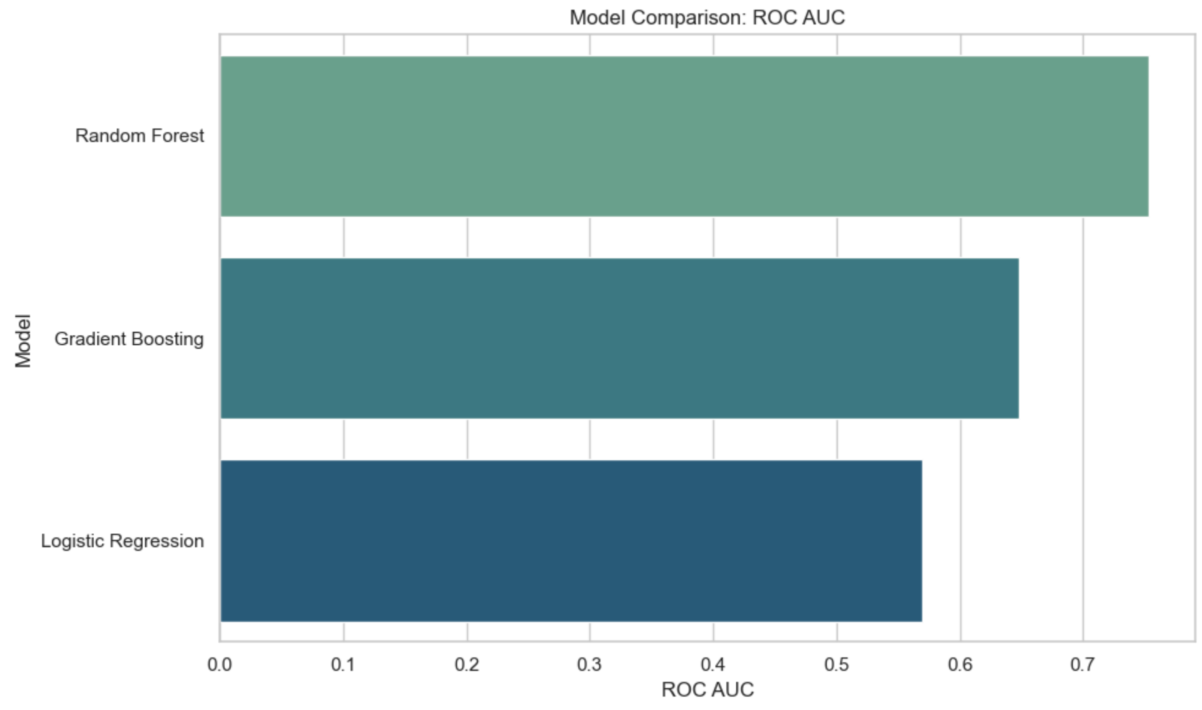
```python
for name, model in models.items():
    print(f"Training: {name}")
    start = time.time()
    model.fit(X_train, y_train)
    end = time.time()

    y_pred = model.predict(X_test)
    y_prob = model.predict_proba(X_test)[:, 1] if hasattr(model, "predict_proba") else model.decision_function(X_test)

    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_prob)

    results.append({
        "Model": name,
        "Accuracy": acc,
        "F1 Score": f1,
        "ROC AUC": auc,
        "Train Time (s)": round(end - start, 2)
    })

# Create results DataFrame
results_df = pd.DataFrame(results)
print(results_df.sort_values(by="F1 Score", ascending=False))

#  Model Comparison Graphs
sorted_results = results_df.sort_values(by="F1 Score", ascending=False)
sns.set(style="whitegrid")
```

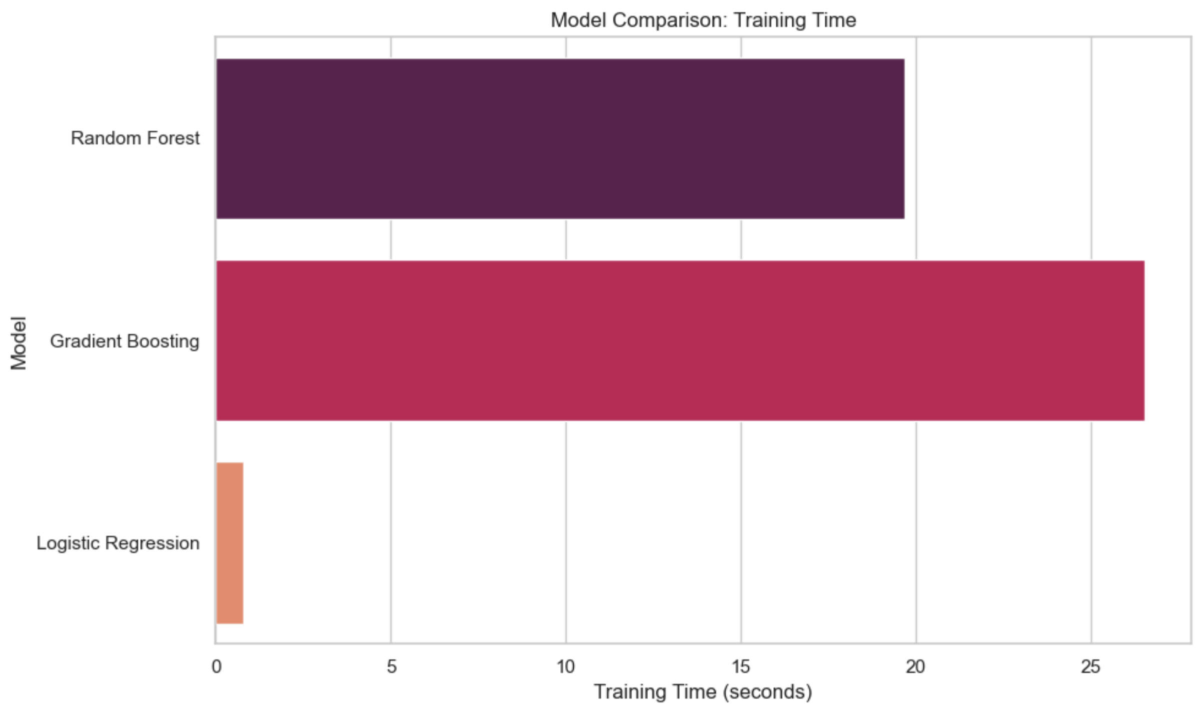
```
# F1 Score
plt.figure(figsize=(10, 6))
sns.barplot(x="F1         Score",        y="Model",        data=sorted_results,
palette="viridis")
plt.title("Model Comparison: F1 Score")
plt.xlabel("F1 Score")
plt.ylabel("Model")
plt.tight_layout()
plt.show()

# ROC AUC
plt.figure(figsize=(10, 6))
sns.barplot(x="ROC AUC", y="Model", data=sorted_results, palette="crest")
plt.title("Model Comparison: ROC AUC")
plt.xlabel("ROC AUC")
plt.ylabel("Model")
plt.tight_layout()
plt.show()

# Training Time
plt.figure(figsize=(10, 6))
sns.barplot(x="Train     Time     (s)",     y="Model",     data=sorted_results,
palette="rocket")
plt.title("Model Comparison: Training Time")
plt.xlabel("Training Time (seconds)")
plt.ylabel("Model")
plt.tight_layout()
plt.show()
```

```
Training: Logistic Regression
Training: Random Forest
```

```
Training: Gradient Boosting
                 Model  Accuracy  F1 Score   ROC AUC  Train Time (s)
1        Random Forest  0.995987  0.224670  0.753467           19.69
2    Gradient Boosting  0.998073  0.175610  0.648299           26.53
0  Logistic Regression  0.603236  0.003436  0.569675            0.79
```







Model Comparison: Training Time
Model
Random Forest
Gradient Boosting
Logistic Regression
0
5
10
15
20
25
Training Time (seconds)