Predictive Modeling with SBA National Data: Insights from the CSU Systemwide Business Analytics Competition

By: Brandon Chang, Darien Lin, & Dylan Ton

Tables of Contents

1 Introduction

This report utilizes historical Small Business Administration (SBA) loan data to develop predictive models aimed at determining the likelihood of loan repayment or default. The dataset consists of 899,164 rows and 27 columns of historical information spanning multiple decades. The primary objective is to assist stakeholders in making informed decisions about loan approvals by maximizing profits and minimizing losses through data-driven insights. By analyzing loan repayment probabilities, the project seeks to identify the most effective classification methods for predicting loan outcomes, incorporating key profitability metrics and risk considerations.

This report will evaluate and compare five different classification models: Logistic Regression, Neural Networks, K-Nearest Neighbors (KNN), Classification Trees, and Discriminant Analysis. These models will be assessed not only based on performance metrics such as accuracy, precision, recall, and F1 score, but also considering profit metrics to ensure that the models contribute to maximizing profitability. The evaluation process will guide key decisions on feature selection, model thresholds, and evaluation metrics to enhance prediction accuracy and optimize business results for stakeholders.

2 Data Exploration

The U.S. Small Business Administration (SBA) supports small businesses through loan guarantees, reducing banks' lending risks while fostering job creation and economic growth. This dataset, based on SBA's National Data, captures historical loan information, including approvals and defaults, to aid in predicting default probabilities. Analyzing this data offers insights into patterns and trends that inform decision-making in the credit market.

Dataset Overview

The dataset used for this analysis consists of 899,164 rows and 27 columns, containing historical Small Business Administration (SBA) loan data.

However, certain columns need to be removed as they represent historical information that would not be applicable for predicting future loan outcomes.

> **Features:** LoanNr_ChkDgt, Name, City, State, Zip, Bank, Bank State, NAICS, Approval Date, ApprovalFY, Term, NoEmp, NewExist, CreateJob, RetainedJob, FranchiseCode, UrbanRural, RevLineCr, LowDoc, ChgOffDate, DisbursementDate, DisbursementGross, BalanceGross, MIS_Status, ChgOffPrinGr, GrAppv, SBA_Appv

Profit/Loss Framework

To evaluate the profitability of loan predictions, the following framework was implemented:

If Predicted Paid in Full, Actual Paid in Full:

- Profit = 5% of DisbursementGross
- This represents the bank earning 5% of the loan amount when a loan is correctly predicted and successfully paid back.

If Predicted Paid in Full, Actual Default:

- Loss = -5 × (5% of DisbursementGross)
- This formula calculates the significant loss incurred when the model incorrectly predicts a loan will be paid in full but it defaults:

If Predicted Default (regardless of the actual outcome):

- Profit/Loss = 0
- This is because the bank chooses not to issue the loan, avoiding both gains and losses.

Key Metrics for Optimizing Profit

False positives cost the bank the most. We need to prioritize limiting false positives.

The following metrics were identified as critical for maximizing profit:

Specificity:

- **Definition:** Specificity measures the model's ability to correctly identify true negatives (cases where loans are predicted to default).
- **Importance:** High specificity reduces false positives, where loans predicted as likely to be paid in full end up defaulting. Avoiding these false positives is crucial to minimizing losses caused by incorrectly approved loans in high-risk scenarios.

Precision:

- **Definition:** Precision evaluates the accuracy of positive predictions (the proportion of correctly predicted positive cases out of all positive predictions).
- **Importance:** High precision ensures that loans predicted as likely to be paid in full are actually successful, reducing the cost of false positives. This directly impacts profitability, as false positives (incorrect approvals) **result in significant losses**.

3 Data Cleaning

One of the first steps in preparing our analysis is to clean the dataset to ensure accuracy and consistency. This involves handling missing or duplicate values, correcting data types, and removing any irrelevant information. Proper data cleaning is essential to improve the quality of insights and prevent errors in subsequent analysis.

1. **Dropped Rows with Null Values**

   We removed rows where the MIS_Status column had null values because it is our dependent variable, and missing values in this column would prevent accurate predictions. Excluding these rows ensures the dataset remains complete and reliable for model training and analysis.

2. **Handled Mixed Data Types**

   The ApprovalFY column contained mixed data types (strings and integers), which could lead to errors in analysis. We cleaned this column by removing the inconsistencies to ensure the data is uniform and can be processed correctly.

3. **Cleaned Monetary Columns**

   The columns DisbursementGross, GrAppv, SBA_Appv, BalanceGross, and ChgOffPrinGr contained dollar signs, commas, and periods, which needed to be removed for accurate numerical analysis. After cleaning these symbols, we converted these columns to integers for consistent data handling.

4. **Remapped NAIC Codes**

   We generalized the NAICS codes by mapping them to their broad industry classification based on the first two digits. This transformation simplifies the data and makes it more meaningful for analysis, especially when working with a large dataset.

5. **Standardized Date Columns**

   The date columns in the dataset were in various formats, which could create inconsistencies during analysis. We standardized these columns by converting them into the appropriate date format to ensure uniformity across the dataset.

6. **Converted to Binary Columns**

   The columns containing 'Y' and 'N' values were converted to binary values (1 and 0) to facilitate numerical analysis. This transformation enables the use of these columns in machine learning models, which typically require numerical input.

7. **Extracted Relevant Time Metric**

We extracted the month from the date columns because time-related features like the month were more relevant for our analysis. This step helps simplify the time-related data and focuses on the most useful information for modeling.

8. **Converted Dependent Variable**

   We converted the MIS_Status column to binary (0 and 1) to align with the requirements of our machine learning model. This binary transformation makes it easier to predict and analyze as a classification problem.

9. **Handled Undefined Values in UrbanRural and NewExist**

   For the UrbanRural and NewExist columns, we replaced the 0 values with "Unknown" since they did not have a defined meaning and represented a significant portion of the data. This change ensures that these values are appropriately handled without skewing the analysis.

10. **Renamed Columns for Readability**

    We remapped the column names to more readable and intuitive labels to make the dataset easier to work with. This step improves the clarity of the dataset and ensures that the columns are self-explanatory for future analysis.

11. **Removed Invalid Entries**

    Columns like Revolving_Line_Of_Credit and Low_Documentation_Loan_Program had random entries like ‘T’, ‘R’, and ‘4’ which were irrelevant and could distort the analysis. We removed these invalid entries to clean the data and maintain its integrity.

12. **Remapped States to Regions**

    We grouped the states into regions to provide a more meaningful representation of geographic data. This change improves the effectiveness of the machine learning model by reducing granularity and focusing on broader geographic patterns.

13. **Dropped Unnecessary Columns**

    We removed columns that were not relevant to the analysis or did not contribute to the model’s performance. This step reduces dataset complexity and ensures that only useful features are included for further analysis.

14. **DataFrames Creation**

    We created three different dataframes: df_new, which retains rows with unknown values; df_new_unknowns, which excludes rows with unknown values; and cleaned_df, which contains additional columns for our neural network model. These different datasets allow us to explore and model the data from multiple perspectives, depending on the analysis.

4 Exploratory Data Analysis

4.1 Features Excluded from the Model

We are excluding certain features from the model to ensure that only relevant and predictive data is used for analysis. By removing features that are too specific, redundant, or only available after loan disbursement, we improve the model's ability to generalize and predict outcomes for new loans. This step also reduces the risk of overfitting, ensuring that the model focuses on factors that are truly indicative of loan success or failure.

1. **Identifiers and Specific Information**

   LoanNr_ChkDgt

   - **Definition**: Unique identifier for each loan.
   - **Reason**: Too specific and irrelevant for prediction. It doesn't contribute to identifying patterns or trends in the data.

   Name

   - **Definition:** Name of the borrower (business).
   - **Reason**: Too specific and could lead to overfitting, as it doesn't generalize across different datasets.

2. **Future Predictions and Irrelevant Historical Data**

   ApprovalFY

   - **Definition**: Fiscal year of loan commitment.
   - **Reason**: Represents a historical feature with irrelevant years, making it unsuitable for predicting future loan trends.

3. **Post-Disbursement Features:**

   CreateJob

   - **Definition:** Number of jobs created from the loan.
   - **Reason**: This information is only available after the loan is disbursed, making it unhelpful for predicting loan approval or status.

   RetainedJob

   - **Definition:** Number of jobs retained due to the loan.
   - **Reason**: This information is only available after the loan is disbursed, which limits its usefulness in prediction models.

ChgOffDate

- **Definition:** Date when the loan defaulted.
- **Reason**: Only applicable to defaulted loans, and irrelevant for predicting outcomes of new loans.

4. **Loan-Specific Financial Data**

BalanceGross

- **Definition:** Outstanding balance of the loan.
- **Reason**: Not applicable to new loans, as no balance exists before the loan is disbursed.

ChgOffPrinGr

- **Definition:** Amount of the principal charged off.
- **Reason:** Only relevant for loans that have defaulted and does not provide useful information for predicting new loans.

GrAppv

- **Definition**: Amount initially approved by the bank.
- **Reason:** Redundant with DisbursementGross, providing no new insights or value to the model.

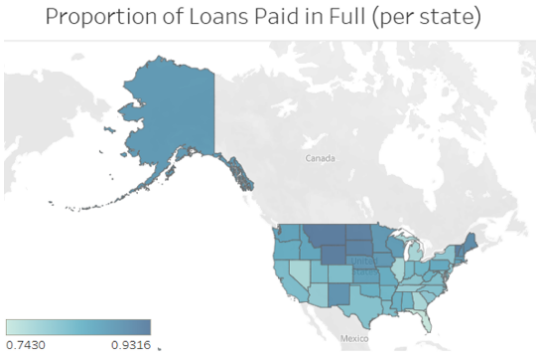
4.2 Data Visualization

In this section, we will explore the relationships between our predictors and the target variable by visualizing how the different classes are distributed across the predictors. By comparing the distributions and trends for each class, we can assess whether the predictors show clear patterns that differentiate the classes, helping us determine which features are likely to be strong predictors for our model. These visual insights will guide us in selecting the most impactful variables for further analysis.

Feature: State

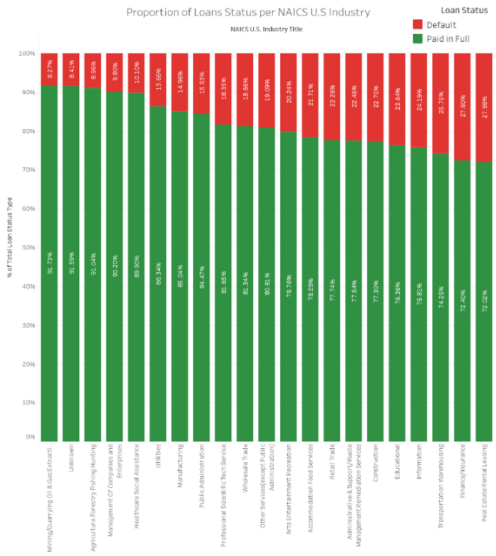
- **Definition:** The U.S. state where the borrower's business is located.
- **Regional Trends:** Some states exhibit higher default rates than others, with noticeable trends across different regions of the United States.
    - **Example:** States like Alaska and parts of the Midwest show higher proportions of loans paid in full (above 90%), indicating stronger repayment reliability.
    - **Contrasting Regions:** States in the South and Southwest, marked by lighter shading, display lower repayment rates (closer to 74%), suggesting higher default risks.

- **Impact on Model:** This feature will be a valuable predictor for our model due to the moderate difference in default/paid-in-full proportions between states. To reduce the risk of overfitting, we can create a new column, 'region,' to group states into broader geographic categories for some of our models.



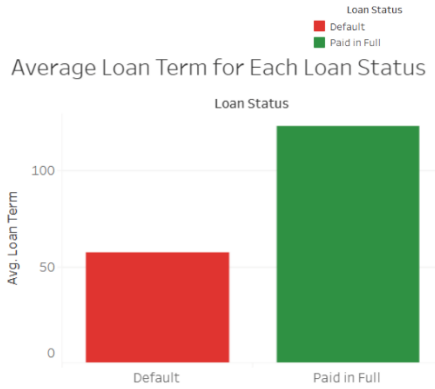
Feature: NAICS U.S. Industry

- **Definition:** The North American Industry Classification System (NAICS) code categorizes the borrower's primary business activity.
- **Industry Trends:** There are moderate to strong differences in default and paid-in-full proportions across NAICS industries, suggesting this feature could be a strong predictor for our model.
    - **Industries with High Repayment:** Sectors such as Mining, Quarrying, Oil and Gas, Unknown, and Agriculture and Forestry are the most likely to pay loans in full, reflecting strong repayment reliability.
    - **Industries with Higher Default Rates:** Sectors like Finance and Real Estate have higher default rates., making them less likely to repay loans in full.
- **Impact on Model:** Given these trends, the NAICS industry code can serve as an important predictor for loan repayment likelihood in our model.



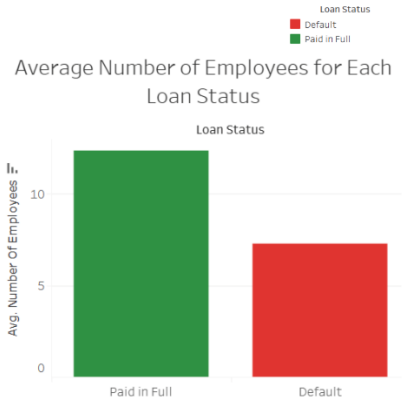
Feature: Loan Term

- **Definition:** The duration of the loan in months, indicating the repayment period agreed upon between the borrower and the lender.
- **Loan Term Trends:** The average loan term for defaulted loans is around 50 months, while paid-in-full loans have an average term of over 100 months.
    - **Longer Loan Terms:** Loans with longer terms are more likely to be paid in full, reflecting a higher likelihood of repayment.
- **Impact on Model:** The loan term will be a strong predictor in the model due to its significant correlation with the likelihood of loan repayment.



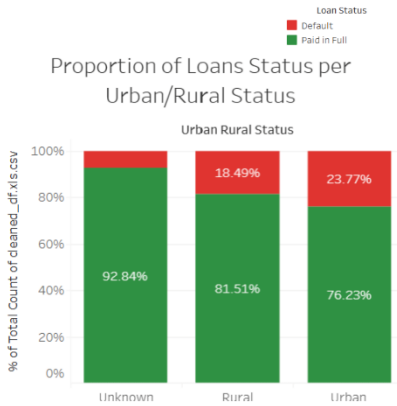
Feature: Number of Employees

- **Definition:** The total number of employees working in the borrower's business.
- **Employee Trends:** There is a notable difference in the average number of employees between paid-in-full loans (approximately 11 employees) and defaulted loans (about 6 employees).
    - **More Employees:** Businesses with a greater number of employees are more likely to pay their loan in full, suggesting better financial stability.
- **Impact on Model:** The number of employees will be a strong predictor in the model due to its significant relationship with loan repayment.



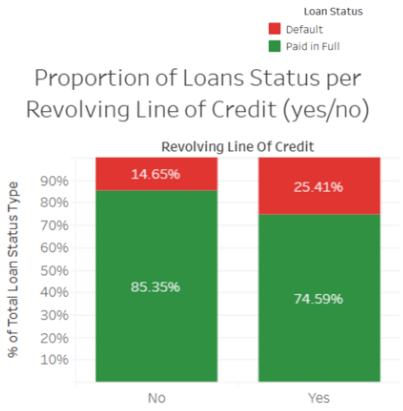
Feature: Urban Rural Status

- **Definition:** Indicates whether the borrower operates in an urban, rural, or unknown area.
- **Urban/Rural Trends:** There are above-moderate differences in the proportion of defaults and paid-in-full loans across the categories of unknown, rural, and urban areas.
    - **Unknown Areas:** Unknown areas have the highest loan repayment rates, followed by rural areas, while urban areas have the highest default rates.
- **Impact on Model:** The urban/rural status will be a strong predictor in the model, as it significantly influences the likelihood of loan repayment across different geographic areas.



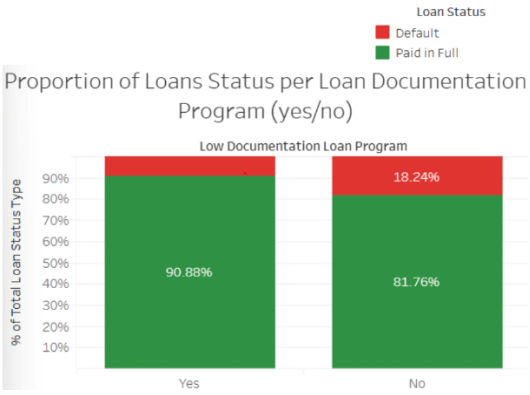
Feature: Revolving Line of Credit

- **Definition:** Indicates whether the loan includes a revolving line of credit, allowing for re-borrowing after repayment.
- **Revolving Line of Credit Trends:** Loans with a revolving line of credit show higher default rates—14.65% defaults for "No" and 25.41% for "Yes."
    - **Higher Default Risk:** Loans with a revolving line of credit are more likely to default.
- **Impact on Model:** The revolving line of credit status will be a strong predictor in the model due to its notable influence on default rates.



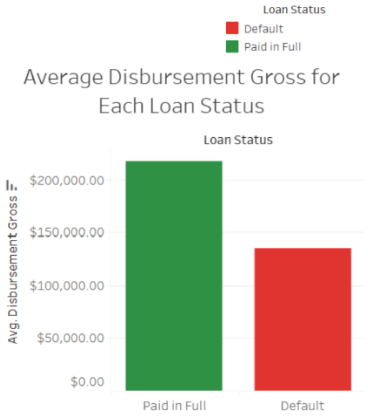
Feature: Low Documentation Loan Program

- **Definition:** Indicates whether the loan was part of the Low Documentation Loan Program, which simplifies the application process by requiring fewer documents.
- **Low Documentation Program Trends:** The Low Documentation Loan Program has a lower default rate for participants (9.12%) compared to non-participants (18.24%).
    - **Lower Default Risk:** Businesses in the Low Documentation Loan Program are less likely to default.
- **Impact on Model:** Participation in the Low Documentation Loan Program will be a valuable predictor in the model, as it shows a significant relationship with default rates.



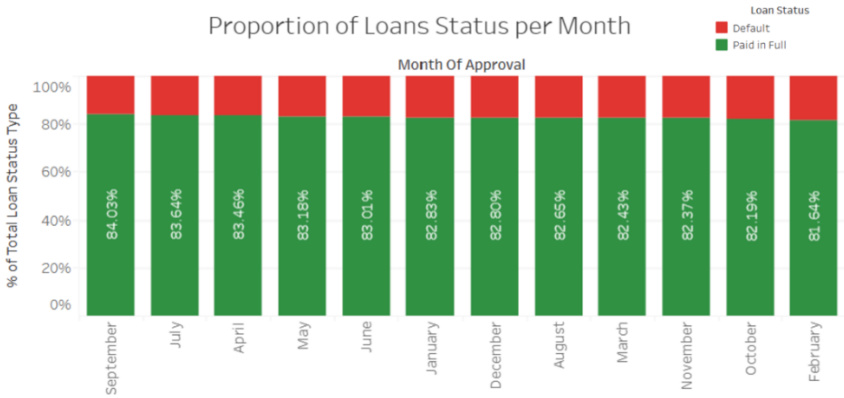
Feature: Disbursement Gross

- **Definition:** The total loan amount disbursed to the borrower.
- **Disbursement Gross Trends:** Loans with higher disbursements are more likely to be paid in full, with paid-in-full loans averaging $210,000 versus $140,000 for defaults.
    - **Higher Loan Amount = Higher Repayment Rate:** A greater loan amount disbursed increases the likelihood of repayment.
- **Impact on Model:** This feature will be a strong predictor as it correlates with loan repayment success.



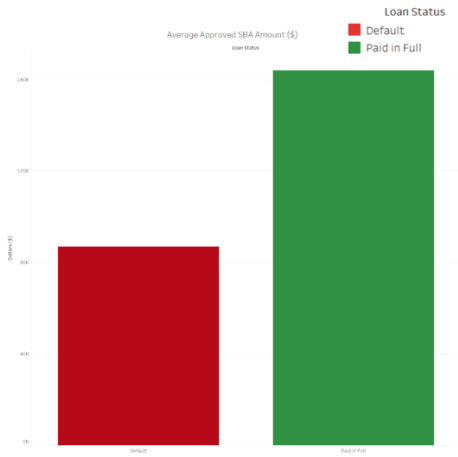
Feature: Month of Approval

- **Definition:** The month in which the loan was approved by the bank.
- **Month of Approval Trends:** The proportion of defaults and paid-in-full loans varies slightly across months, with paid-in-full loans ranging from 81.64% to 84.03%.
    - **Minimal Impact on Loan Repayment:** The month of approval shows only small differences in repayment rates, indicating it will likely be a weaker predictor.
- **Impact on Model:** This feature is expected to have limited predictive power due to its minimal correlation with loan outcomes.



Feature: SBA Approved Amount

- **Definition:** The portion of the loan guaranteed by the Small Business Administration (SBA), representing the amount the SBA commits to covering if the borrower defaults.
- **SBA Approved Amount Trends:** Higher SBA-approved amounts are linked to full repayments, with defaults averaging $80,000 and paid-in-full loans $160,000.
    - **Impact on Repayment:** A higher SBA-approved amount is correlated with a greater likelihood of the loan being paid in full.
- **Impact on Model:** This feature is a strong predictor but may correlate with 'Disbursement Gross.' To avoid multicollinearity, we may retain one in the models.



5 Methodology & Results

5.1 Discriminant Analysis

In this section, we will test both Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA). LDA assumes that the predictors have a linear relationship, while QDA allows for non-linear relationships between the variables. By comparing both methods, we aim to determine which approach better classifies the data based on the features available.

Feature Engineering

In the feature engineering process, we used different datasets for Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA). For QDA, we used fewer features by dropping 'Month_Of_Approval', 'NAICS_U.S._Industry_Title', and 'Franchise_Status' to reduce complexity and mitigate the risk of overfitting. These features were removed for the following reasons:

- **'Month_Of_Approval'** showed little relevance to the outcome, potentially introducing noise, as the month of approval doesn't have a clear or consistent relationship with loan defaults or repayments.
- **'NAICS_U.S._Industry_Title'** was a categorical variable with high cardinality, which could complicate the non-linear modeling process. Additionally, this feature did not provide significant discriminatory power for QDA, making it less useful for classification.
- **'Franchise_Status'** had limited variance across the dataset, offering minimal discriminatory power for QDA, as most loans either were or weren't from franchises, providing little distinction between the classes.

By removing these features for QDA, we aimed to improve model performance, reduce overfitting, and ensure that only the most relevant features were used for classification, allowing QDA to better capture the complex relationships in the data.

Training & Validation

For the training and validation process, we performed an 80/20 split of the data, using 80% for training and 20% for testing. This resulted in over 170,000 data points being allocated for testing, ensuring a robust evaluation of model performance. This split allows for adequate training of the models while providing a substantial test set to validate their generalizability.

Additionally, we applied one-hot encoding to categorical features, using the drop_first=True parameter to avoid multicollinearity. This resulted in binary columns for each category, while

dropping the first category in each feature to serve as the reference category, which is a common practice to maintain model interpretability and reduce redundancy.

After the data split, we normalized the features to ensure consistency in the scale of the input variables, which helps improve model performance by preventing features with larger scales from dominating the learning process. This normalization step was performed separately on the training and testing sets to avoid data leakage and ensure the integrity of the validation process.

Hyperparameters

We did not have any predefined hyperparameters for the models, but we experimented with some adjustments to improve performance. For example, we tried adding class weights to address class imbalances, but this did not yield favorable results.

- **Specificity Increase:** Specificity improved notably, with values rising from 0.9782 (Training) to 0.9763 (Test).
- **Decline in Other Metrics:** While specificity improved, key metrics such as accuracy, precision, recall, and F1 score saw a sharp decline, indicating that these adjustments did not enhance the overall model performance.
- **Impact on Model Generalization:** The adjustments, while improving specificity, led to a reduction in generalization ability, as seen in the deterioration of other important metrics.

This suggests that class weighting may have overcorrected the model's focus on the minority class at the expense of other performance indicators.

Metrics

Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA), we evaluated the model's performance on both the training and test datasets across various metrics. Below are the training and test metrics for LDA, providing insight into its effectiveness and generalization ability.

Training Set Metrics:	Test Set Metrics:
Accuracy: 0.8327	Accuracy: 0.8329
Precision: 0.8176	Precision: 0.8191
Recall: 0.8327	Recall: 0.8329
F1 Score: 0.7640	F1 Score: 0.7639
Specificity: 0.0345	Specificity: 0.0333

The Linear Discriminant Analysis (LDA) model had strong metrics for accuracy, precision, recall, and F1 score, but low specificity, indicating challenges in correctly classifying negative cases. Below are the training and test metrics for LDA.

Quadratic Discriminant Analysis (QDA)

For Quadratic Discriminant Analysis (QDA), we evaluated the model's performance after dropping three features: 'Month_Of_Approval', 'NAICS_U.S._Industry_Title', and 'Franchise_Status'. These features were removed to simplify the model and reduce the risk of overfitting. Below are the training and test metrics for QDA, which provide insights into its performance after this feature engineering adjustment.

```
Training Set Metrics:  Test Set Metrics:
Accuracy: 0.6448       Accuracy: 0.6460
Precision: 0.8144      Precision: 0.8156
Recall: 0.6448         Recall: 0.6460
F1 Score: 0.6885       F1 Score: 0.6896
Specificity: 0.7465    Specificity: 0.7494
```

The resulting metrics are significantly better, with notably higher specificity compared to LDA, indicating that QDA performs better in correctly identifying negative cases. Below are the training and test metrics for QDA.

Cross Validation Folds

After performing K-Fold cross-validation with Quadratic Discriminant Analysis (QDA), the metrics (accuracy, precision, recall, F1 score, and specificity) were consistent across all folds. This indicates that the model generalizes well to unseen data, demonstrating no signs of overfitting.

```
Metrics for each fold:
Fold 1 - Accuracy: 0.6614, Precision: 0.9245, Recall: 0.6441, F1 Score: 0.7592, Specificity: 0.7451
Fold 2 - Accuracy: 0.6216, Precision: 0.9219, Recall: 0.5938, F1 Score: 0.7223, Specificity: 0.7563
Fold 3 - Accuracy: 0.6464, Precision: 0.9217, Recall: 0.6266, F1 Score: 0.7460, Specificity: 0.7421
Fold 4 - Accuracy: 0.6459, Precision: 0.9236, Recall: 0.6244, F1 Score: 0.7451, Specificity: 0.7497
Fold 5 - Accuracy: 0.6575, Precision: 0.9239, Recall: 0.6394, F1 Score: 0.7558, Specificity: 0.7450

Average Metrics Across All Folds:
Average Accuracy: 0.6466
Average Precision: 0.9231
Average Recall: 0.6257
Average F1 Score: 0.7457
Average Specificity: 0.7476
```

Profit

Linear Discriminant Analysis (LDA)

Below are the profit metrics for Linear Discriminant Analysis (LDA). These figures represent the model's effectiveness in predicting outcomes and the associated profit based on its predictions.

```
Total Net Profit/Loss: 578984058.50
Average Total Net Profit/Loss: 3316.36
```

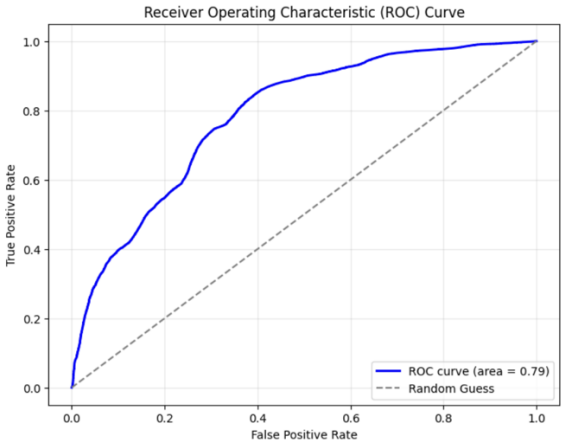
Quadratic Discriminant Analysis (QDA)

Below is the profit for Quadratic Discriminant Analysis (QDA), which is much higher than LDA, reflecting its improved performance and profitability.

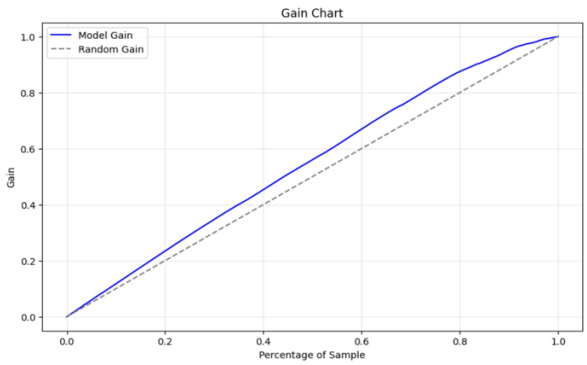
```
Total Net Profit/Loss: 786170263.75
Average Total Net Profit/Loss: 4503.11
```

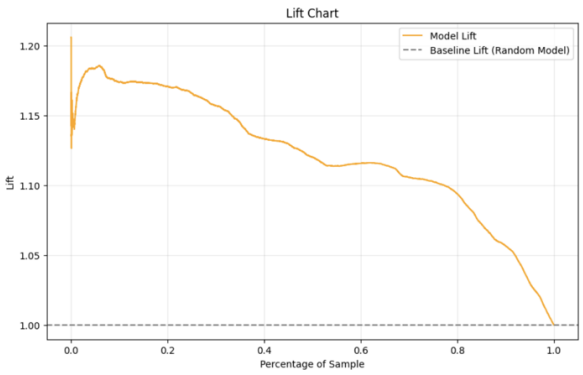
AUC-ROC

Since QDA performed better, we will show the AUC curve, where it achieved a score of 0.79, further demonstrating its effectiveness in distinguishing between classes.



Gain & Lift Chart





The gain chart shows that QDA will perform better than random, but it lacks strong predictive power, indicating that while it offers improvement over random chance, there is still room for better model performance. The lift chart shows subpar performance in the lower deciles, indicating that QDA struggles to provide significant improvements in predictions for the lower range of the data.

5.2 Logistic Regression

In this section, we will also be exploring Logistic Regression, along with regularized models including Lasso, Ridge, and ElasticNet. These techniques will help assess the impact of regularization on model performance and improve our ability to handle potential overfitting.

Feature Engineering

For the feature engineering in Logistic Regression, we kept the dataset after data cleaning without dropping any columns. This choice was because all features were relevant and provided valuable insights for predicting loan defaults. Since Logistic Regression is less prone to overfitting compared to more complex models, we retained all the features to ensure the model had a comprehensive set of predictors, reflecting the cleaned data's full structure.

Training & Validation

For training and validation, we used a 70/30 split, with around 260,000 samples in the test set. This ensured a large test set for evaluation while retaining enough data for training. After creating dummy variables, we normalized the data to ensure consistent scaling, performing normalization post-split to avoid test set leakage.

Hyperparameters

These three models—Logistic Regression, Lasso Regression, and ElasticNet—were the best-performing models in our analysis. The hyperparameters for each model were as follows:

- **Logistic Regression:** We applied a standard Logistic Regression model with a maximum of 1000 iterations and class balancing to address class imbalance.
- **Lasso Regression:** We used Lasso regularization by setting the penalty to 'l1', along with the 'liblinear' solver. This configuration allows for feature selection by shrinking some coefficients to zero, also with class balancing and 1000 iterations.
- **ElasticNet:** This model combined both Lasso and Ridge regularization by setting the penalty to 'elasticnet' and the l1_ratio to 0.5. The solver was set to 'saga', which is efficient for large datasets. We also included class balancing and 1000 iterations.

Additionally, we experimented with SMOTE (Synthetic Minority Over-sampling Technique) to balance the classes by generating synthetic samples. However, this led to overfitting, as the models struggled to generalize effectively due to the inflated number of minority class samples.

Metrics

Logistic Regression

Below are the Logistic Regression metrics, which show good precision of 0.95 and specificity of 0.8251. Additionally, the model has a runtime of 2.89 seconds, indicating efficient performance.

```
Training Set Metrics:     Test Set Metrics:
Accuracy: 0.7066          Accuracy: 0.7076
Precision: 0.9497         Precision: 0.9500
Recall: 0.6820            Recall: 0.6834
F1 Score: 0.7939          F1 Score: 0.7950
Specificity: 0.8254       Specificity: 0.8251
```

Lasso Regression

Lasso Regression demonstrates strong metrics with a specificity of 0.8251, precision of 0.95, and a runtime of 4.28 seconds, reflecting its effective performance in the model.

```
Training Set Metrics:  Test Set Metrics:
Accuracy: 0.7065       Accuracy: 0.7075
Precision: 0.9497      Precision: 0.9500
Recall: 0.6820         Recall: 0.6834
F1 Score: 0.7939       F1 Score: 0.7949
Specificity: 0.8252    Specificity: 0.8251
```

ElasticNet

ElasticNet had similar metrics with a specificity of 0.8251 and precision of 0.95, but its runtime of 300.39 seconds may not be optimal for practical use.

```
Training Set Metrics:  Test Set Metrics:
Accuracy: 0.7065       Accuracy: 0.7076
Precision: 0.9497      Precision: 0.9500
Recall: 0.6820         Recall: 0.6834
F1 Score: 0.7939       F1 Score: 0.7949
Specificity: 0.8252    Specificity: 0.8251
```

Cross Validation Folds

We performed cross-validation on the models, and the results were consistent across all folds, with metrics such as precision, specificity, and accuracy remaining stable. We focused on Logistic Regression and Lasso Regression because they offer a good balance of performance and interpretability, which is crucial for the bank's decision-making process. These models are more practical for real-world implementation, as they are computationally efficient, easy to deploy, and provide clear insights that can assist in the loan approval process. Given these advantages, we chose not to pursue more complex models like ElasticNet, despite its similar performance, due to its high runtime.

Logistic Regression

```
Metrics for each fold:
Fold 1 - Precision: 0.9505, Recall: 0.683, F1 Score: 0.7948, Accuracy: 0.7077, Specificity: 0.8277
Fold 2 - Precision: 0.949, Recall: 0.6802, F1 Score: 0.7924, Accuracy: 0.7046, Specificity: 0.8231
Fold 3 - Precision: 0.9492, Recall: 0.6834, F1 Score: 0.7946, Accuracy: 0.7073, Specificity: 0.8229
Fold 4 - Precision: 0.9509, Recall: 0.6784, F1 Score: 0.7919, Accuracy: 0.7044, Specificity: 0.8304
Fold 5 - Precision: 0.9492, Recall: 0.6808, F1 Score: 0.7929, Accuracy: 0.7053, Specificity: 0.8236

Average Metrics Across All Folds:
Average Accuracy: 0.7059
Average Precision: 0.9498
Average Recall: 0.6812
Average F1 Score: 0.7933
Average Specificity: 0.8255
```

Lasso Regression

```
Metrics for each fold:
Fold 1 - Precision: 0.9504, Recall: 0.6828, F1 Score: 0.7947, Accuracy: 0.7076, Specificity: 0.8276
Fold 2 - Precision: 0.949, Recall: 0.68, F1 Score: 0.7923, Accuracy: 0.7045, Specificity: 0.823
Fold 3 - Precision: 0.9493, Recall: 0.6833, F1 Score: 0.7946, Accuracy: 0.7072, Specificity: 0.8231
Fold 4 - Precision: 0.9508, Recall: 0.6785, F1 Score: 0.7919, Accuracy: 0.7044, Specificity: 0.8298
Fold 5 - Precision: 0.9493, Recall: 0.681, F1 Score: 0.7931, Accuracy: 0.7055, Specificity: 0.8237

Average Metrics Across All Folds:
Average Accuracy: 0.7058
Average Precision: 0.9497
Average Recall: 0.6811
Average F1 Score: 0.7933
Average Specificity: 0.8254
```

Profit

The profits for both Lasso and Logistic Regression were similar, suggesting that both models provided comparable financial performance.

Logistic Regression

```
Total Net Profit/Loss: 1310858367.60
Average Total Net Profit/Loss: 5005.65
```

Lasso Regression

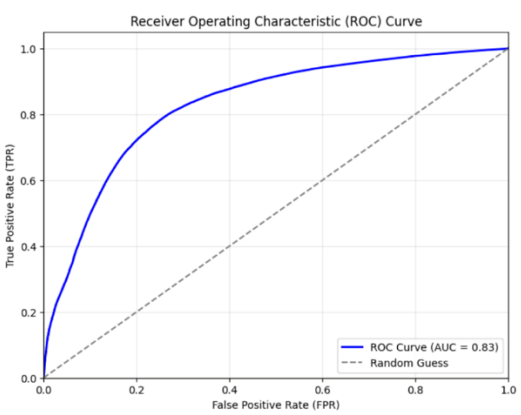
```
Total Net Profit/Loss: 1310786593.55
Average Total Net Profit/Loss: 5005.37
```

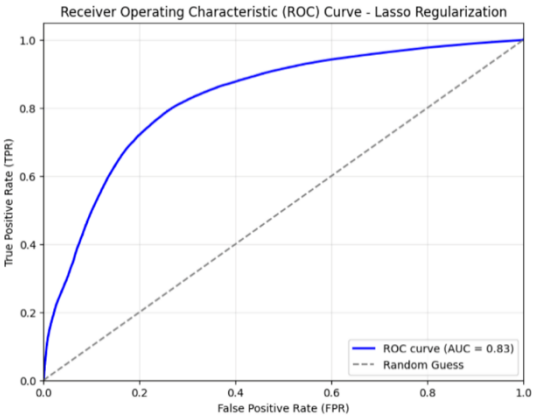
This similarity in profit highlights that, despite some differences in their complexity and training times, both models are effective at predicting loan defaults and repayments, making them suitable for practical use in the bank's decision-making process.

AUC-ROC

Below are the ROC curves for both models: on the right is Lasso Regression and on the left is

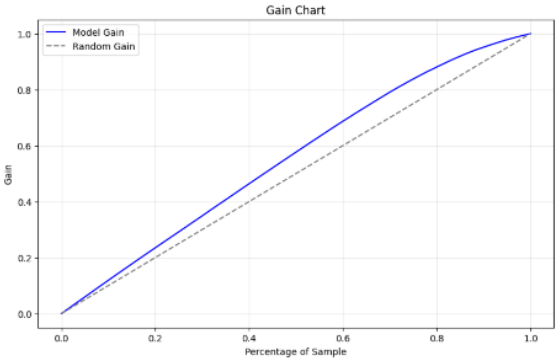
Logistic Regression. Both models achieved an AUC of 0.83, demonstrating similar performance in distinguishing between defaulted and paid-in-full loans.

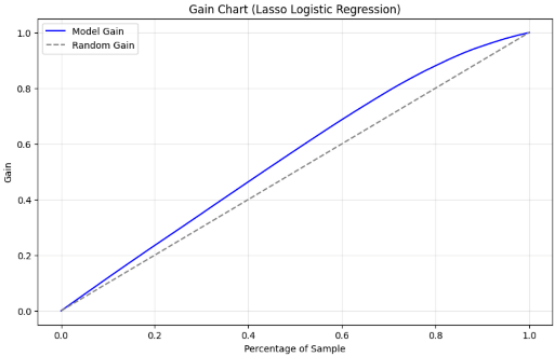


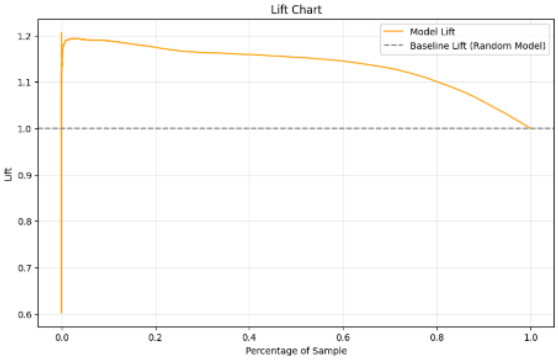


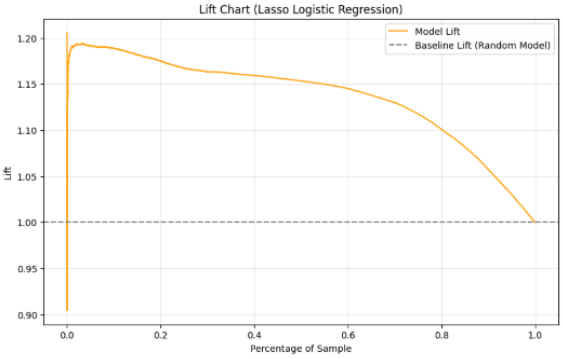
Gain & Lift Chart

Below are the Gain and Lift charts, with Logistic Regression on the left and Lasso Regression on the right. Both charts show similar performance, with both models outperforming random selection, indicating their effectiveness in identifying high-probability loan defaults.









5.3 K-Nearest Neighbors (KNN)

In this section, we explore the performance of the k-Nearest Neighbors (KNN) model for predicting loan defaults. KNN is a non-parametric algorithm that makes predictions based on the proximity of data points in the feature space. We will evaluate its effectiveness and compare it with the other models used in this analysis.

Feature Engineering

Since KNN is computationally intensive, running it on the entire dataset of 800,000 rows would be impractical. To address this, we sampled a subset of 100,000 rows, enabling us to efficiently adjust and optimize the model. Additionally, we dropped columns likely to introduce noise, based on frequency visualizations that revealed limited variability or inconsistent relationships with the target variable.

The following columns were removed:

- **Urban_Rural_Status:** Minimal variation in values reduced its utility for classification.
- **Region:** Provided limited discriminatory power, as its distribution didn't significantly correlate with loan outcomes.
- **Month_Of_Approval:** Showed no consistent relationship with the likelihood of defaults, making it non-informative for KNN.

These steps allowed us to refine the dataset and enhance the model's performance while maintaining computational feasibility.

Training & Validation

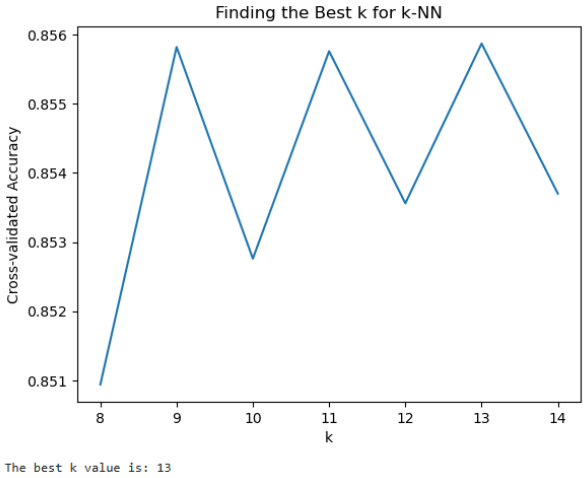
We used an 80/20 training and validation split, providing a robust balance by allowing the model to train on a significant portion of the data while retaining approximately 20,000 validation rows for statistically reliable evaluation of the model's performance.

Numerical features were standardized using Z-score normalization, ensuring all features had a mean of 0 and a standard deviation of 1. This step was critical for K-Nearest Neighbors (KNN), a distance-based algorithm, to prevent features with larger ranges from disproportionately influencing distance calculations. For categorical variables, we created dummy variables, converting them into numerical formats compatible with KNN and ensuring the model could effectively interpret categorical data.

Hyperparameters

For the hyperparameters in our KNN model, we focused on determining the optimal number of neighbors (k) to balance bias and variance effectively. To achieve this, we implemented a function that systematically evaluated the model's performance across multiple values of k within the range of 8 to 15. This range was selected based on prior experimentation and domain knowledge, aiming to capture the optimal balance point.

The function computed the mean metrics for each k value, including accuracy, precision, recall, F1 score, and specificity, ensuring a comprehensive evaluation of performance. After analyzing the results, k = 13 emerged as the best value, providing the highest overall performance across our metrics. This value struck the ideal balance by minimizing noise while maintaining sufficient local information for robust classification.



Additionally, we tested different distance metrics (e.g., Euclidean and Manhattan distances) but found Euclidean distance to deliver the best results, aligning well with our standardized data. These hyperparameter choices aimed to maximize the predictive power of the KNN model while keeping computational complexity manageable.

Metrics

We tested the KNN model using both oversampled and regular datasets to evaluate its performance. The regular dataset achieved a high precision of 0.9 but had a low specificity of 0.4841, indicating that the model struggled to correctly identify negative cases.

```
Training Set Accuracy: 0.88          Testing Set Accuracy: 0.87
Training Set Precision: 0.90         Testing Set Precision: 0.89
Training Set Recall: 0.96            Testing Set Recall: 0.95
Training Set F1 Score: 0.93          Testing Set F1 Score: 0.92
Specificity (Training Set): 0.4841   Specificity (Test Set): 0.4407
```

When using oversampling techniques, the metrics initially appeared improved, with the testing specificity rising to 0.9167 and the testing precision reaching 0.7685. However, these results revealed signs of overfitting, as the model performed disproportionately well on the training set but lacked generalization to unseen data. Due to this overfitting, we determined that the oversampled approach was not suitable for our KNN implementation.

```
Training Set Accuracy: 0.87          Testing Set Accuracy: 0.80
Training Set Precision: 0.91         Testing Set Precision: 0.95
Training Set Recall: 0.83            Testing Set Recall: 0.81
Training Set F1 Score: 0.87          Testing Set F1 Score: 0.87
Specificity (Training Set): 0.9167   Specificity (Test Set): 0.7685
```

Profit

The average net profit for KNN without oversampling was $4,530.85, showing positive returns but not enough to offset the trade-offs in performance, especially the balance between precision and specificity.

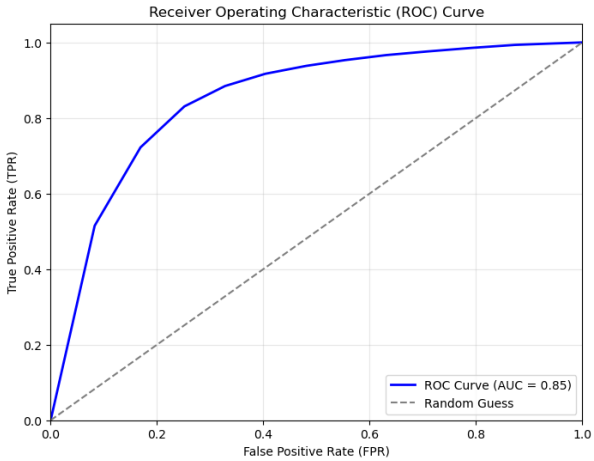
```
Total Net Profit/Loss: 90617041.44999999
Average Total Net Profit/Loss: 4530.8520725
```

The KNN model with oversampling yielded a profit of $5,196.30, slightly higher than without oversampling. However, due to significant overfitting, we opted for the more reliable KNN model without oversampling.

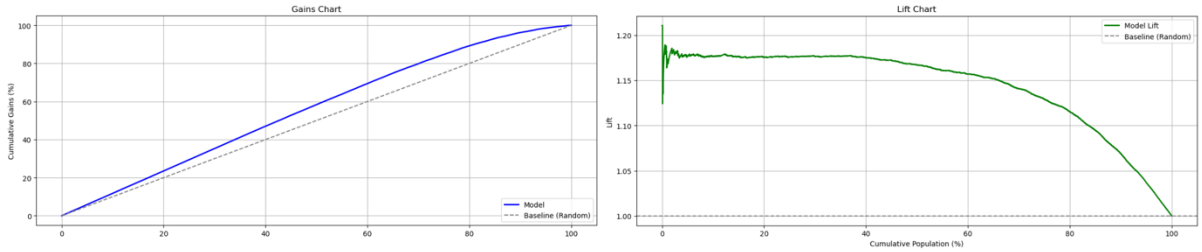
```
Total Net Profit/Loss: 103926142.75
Average Total Net Profit/Loss: 5196.3071375
```

AUC-ROC

The AUC of 0.85 indicates that the model performs moderately well in distinguishing between paying and non-paying businesses. This suggests the model is reasonably reliable for predicting whether a business will fully repay its loan.



Gain & Lift Chart



The model outperforms random but struggles to effectively prioritize positives in the early deciles, likely due to the absence of charge-off data, which limits the ability to assess its true performance through the gains chart, making it less informative for evaluation.

5.4 Neural Networks

In this section, we will explore the use of neural networks to model the data and improve predictive performance. Neural networks offer a flexible approach to capturing complex, non-linear relationships that may not be fully addressed by traditional models. We will discuss the architecture, hyperparameters, and evaluation of the neural network model used in this analysis.

Feature Engineering

Our feature engineering process retained a significant portion of the original dataset's features, leveraging the strengths of neural networks in capturing non-linear relationships. Unlike linear models, such as logistic regression, which heavily depend on eliminating multicollinearity and reducing categorical complexity to prevent overfitting, neural networks can effectively learn from high-dimensional data and subtle patterns. As a result, we did not prioritize removing multicollinearity, as neural networks are robust in handling correlated features. Similarly, features with numerous categories were retained because neural networks can encode and learn from these variables, even if there is no strong linear correlation between them and the target variable.

Key decisions in the feature engineering process included:

- **Retaining Categorical Features:** Variables such as the borrower's state of operation (State), industry classification (NAICS_U.S._Industry_Title), business type (Business_Type), urban/rural status (Urban_Rural_Status), franchise status (Franchise_Status), and the month of loan approval (Month_Of_Approval) were preserved, as neural networks excel at learning from high-cardinality categorical variables.

- **Numerical Features**: Key numerical features, such as loan term (Loan_Term), number of employees (Number_Of_Employees), disbursed loan amount (Disbursement_Gross), SBA-approved loan amount (SBA_Approved_Amount), revolving credit line status (Revolving_Line_Of_Credit), and LowDoc loan program participation (Low_Documentation_Loan_Program), were also retained for their potential contribution to model learning.
- **Handling Correlations**: Even with weak correlations between some features and the target, neural networks are capable of identifying complex, non-linear interactions that linear models may miss.

By maintaining these features, we enabled the neural network to leverage its flexibility in learning intricate relationships and improving predictive accuracy without significant feature reduction. This approach demonstrates the advantage of neural networks in handling complex datasets with diverse, high-dimensional features, highlighting their ability to capture interactions and patterns beyond the reach of traditional linear models.

Training & Validation

We used an 80/20 training and validation split, providing a balanced approach that allowed the model to train on a substantial portion of the data while retaining around 170,000 validation rows for reliable and statistically significant performance evaluation. This split ensured that the model had sufficient data to learn from while also maintaining enough data to assess its generalization capabilities effectively.

For data preparation, we normalized the numerical values using Z-score standardization, ensuring all features were scaled with a mean of 0 and a standard deviation of 1. This normalization step is important for the MLP neural network, as it helps the model converge more quickly, improves numerical stability during training, and ensures that no feature dominates the learning process due to its scale. Additionally, we converted categorical variables into dummy variables to make them compatible with the neural network. Unlike traditional models, we did not drop the first category when creating dummies, as neural networks are capable of handling multicollinearity effectively, allowing them to learn from redundant information without negatively impacting performance.

Hyperparameters

Our neural network training process involved careful consideration of various activation functions, loss functions, and optimization strategies to identify the best combination for robust performance. After experimenting with multiple options, we selected the ReLU activation function, Cross-Entropy Loss, and the Adam optimizer, as they demonstrated superior performance and generalization capabilities. Here, we discuss the other functions and optimizers explored and explain why the selected options outperformed them.

Activation Functions: ReLU vs. Alternatives

We tested several activation functions to find the best fit for our neural network:

- **Sigmoid:** Compresses inputs into the range (0, 1), which causes gradients to vanish as inputs approach these limits. This leads to the vanishing gradient problem, slowing down weight updates in deeper networks. ReLU is preferred because it avoids the vanishing gradient problem and is computationally efficient, enabling faster training.
- **Tanh:** Outputs values between (-1, 1), which addresses the zero-centering issue found in sigmoid but still suffers from the vanishing gradient problem for larger inputs. It is also more computationally intensive compared to ReLU. ReLU is preferred for its simplicity, efficiency, and faster training time.
- **Leaky ReLU:** solves the "dying ReLU" problem by giving negative inputs a small slope, ensuring some gradient flows even for negative values. Despite this advantage, it didn't show a significant improvement over ReLU for our dataset. ReLU was preferred due to its simplicity and better performance without added complexity.
- **Softmax:** Is typically used in the output layer for multi-class classification and not in hidden layers. It generates class probabilities but is less suited for hidden layers compared to ReLU. ReLU was used in the hidden layers, while softmax was incorporated only in the output layer for class prediction.

Loss Functions: Cross-Entropy vs. Alternatives

We explored several loss functions to determine the most effective one for our classification tasks:

- **Mean Squared Error (MSE):** MSE penalizes all errors equally, which is not ideal for classification tasks where probabilistic outputs matter. It leads to slower convergence and less accurate decision boundaries for classification problems. Cross-Entropy is preferred as it measures the divergence between predicted probabilities and true labels, resulting in faster and more effective learning.
- **Hinge Loss:** Is designed for binary classification with support vector machines (SVMs) and doesn't handle probabilistic outputs well. It is not suitable for tasks that require class probabilities, such as neural networks. Cross-Entropy is preferred because it optimizes for probabilistic outputs, making it better suited for classification tasks.
- **Huber Loss:** Is primarily used for regression tasks and doesn't effectively distinguish between classes in classification tasks. It is less suited for optimizing classification accuracy. Cross-Entropy is better for classification because it directly minimizes classification error while capturing probabilistic outputs.
- **KL Divergence:** Measures the difference between two probability distributions but assumes one distribution as a reference, making it less flexible for general classification tasks. It is not ideal for tasks where both distributions are important to compare. Cross-

Entropy is better because it minimizes the divergence between predicted and true distributions without needing a fixed reference.

Optimization: Adam vs. Alternatives

We tested several optimization algorithms to determine the most effective one for updating model weights:

- **Stochastic Gradient Descent (SGD):** SGD uses a fixed learning rate, making it less efficient in adapting to different gradients, and can suffer from oscillations, especially in non-convex landscapes. It struggles to converge effectively in such scenarios. Adam is preferred as it adapts the learning rate dynamically, leading to faster and more stable convergence.
- **Momentum:** Accelerates convergence by adding a fraction of the previous gradient to the current update, which reduces oscillations. It helps smooth the learning process and speeds up convergence. Adam is better because it incorporates momentum while using RMSProp's adaptive learning rates, offering a combination of benefits.
- **RMSProp:** Adjusts learning rates for each parameter based on the average of recent gradients, improving convergence in non-convex loss landscapes. It helps stabilize updates by adapting the learning rate. Adam is preferred as it extends RMSProp by incorporating momentum, resulting in smoother and faster updates.
- **Adagrad:** Adagrad works well in sparse settings but reduces learning rates over time, which can hinder long-term training performance. As the learning rate diminishes, the model becomes less effective over time. Adam avoids this issue by using moving averages for gradients and squared gradients, preventing the learning rate from diminishing too quickly.

Hyperparameters & Model Design

The selection of hyperparameters was carefully optimized to ensure efficient training, robust generalization, and avoidance of overfitting. Key choices included a three-layer network architecture, appropriate neuron counts in each layer, and optimizers like Adam, which dynamically adjusts learning rates. Together, these decisions ensured that the model balanced learning complexity with computational efficiency, leading to strong predictive performance.

- **Neural Network Architecture:** The network was designed with three hidden layers, with 64, 48, and 32 neurons in the first, second, and third layers, respectively. This architecture captured complex patterns while avoiding overfitting and excessive computational cost. The gradual reduction of neurons between layers helped the network process information effectively, improving both efficiency and performance.
- **Learning Rate and Optimizer:** We used a learning rate of 0.001 with the Adam optimizer, which dynamically adjusts learning rates for each parameter. This choice provided a good trade-off between stability and convergence speed, ensuring efficient training without overshooting the optimal loss. Adam's adaptive learning rate minimized the need for extensive manual tuning.

- **Batch Size:** A batch size of 32 was chosen to balance computational efficiency and frequent weight updates, contributing to faster convergence. This batch size was also optimal for GPU memory limits, ensuring smooth and uninterrupted training. It allowed for regular weight adjustments without overloading computational resources.
- **Class Weighting:** To address class imbalance, class weights of [2.5, 0.5] were applied, giving more importance to the minority class. This adjustment helped the model focus on harder-to-classify samples while preventing the majority class from dominating the predictions. The weights ensured balanced model performance while improving classification accuracy for the minority class.
- **Early Stopping and Epoch Control:** Early stopping with a patience of 3 epochs and a minimum improvement threshold of 0.05 was used to prevent overfitting and unnecessary computation. This strategy allowed training to continue only when meaningful improvements were made in validation loss, ensuring efficient use of resources. The patience value provided flexibility for the model to make gains without prematurely halting training.

The hyperparameter choices, including the network architecture, learning rate, batch size, class weights, and early stopping criteria, were all carefully calibrated to optimize model performance. These settings allowed the network to achieve strong predictive capabilities while ensuring generalization and preventing overfitting

Metrics

The model performs well in predicting loan outcomes, with training and validation accuracies of 87.31% and 86.92%, respectively, indicating strong performance and minimal overfitting. High precision (~0.96) reduces false positives, critical for profitability, while high recall (~0.87) ensures most true positives are captured. Specificity (~0.85) minimizes misclassifications of paid-in-full loans, further enhancing profitability.

```
Training Metrics:              Validation Metrics:
Training Accuracy: 0.8731      Validation Accuracy: 0.8692
Training Precision: 0.9672     Validation Precision: 0.9641
Training Recall: 0.8766        Validation Recall: 0.8751
Training F1 Score: 0.9197      Validation F1 Score: 0.9174
Training Specificity: 0.8562   Validation Specificity: 0.8404
```

Cross Validation Folds

Cross-validation results confirm the model's consistency, with an average accuracy of 87.31%, precision of 0.9672, and F1 score of 0.9197, demonstrating effective generalization and accurate loan predictions.

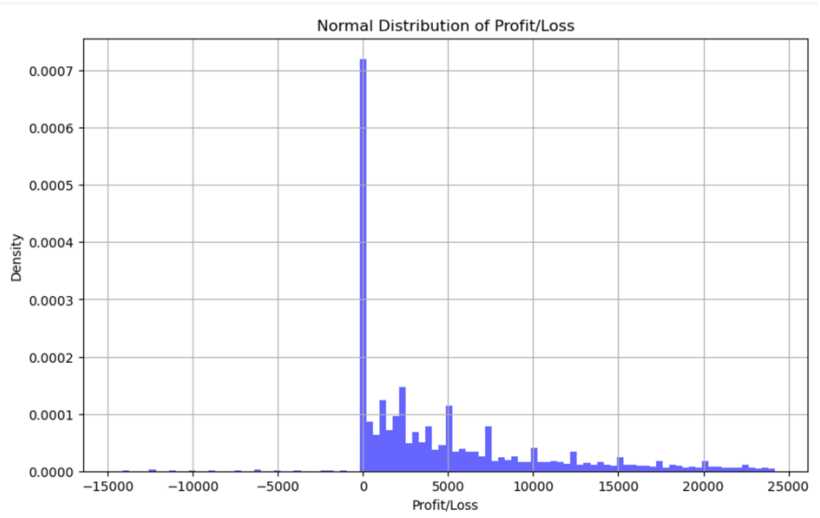
```
Fold 1
Fold 1 Metrics:
Accuracy: 0.8739, Precision: 0.9670, Recall: 0.8776, F1 Score: 0.9201, Specificity: 0.8559
Fold 2
Fold 2 Metrics:
Accuracy: 0.8733, Precision: 0.9676, Recall: 0.8764, F1 Score: 0.9198, Specificity: 0.8578
Fold 3
Fold 3 Metrics:
Accuracy: 0.8730, Precision: 0.9668, Recall: 0.8766, F1 Score: 0.9195, Specificity: 0.8559
Fold 4
Fold 4 Metrics:
Accuracy: 0.8738, Precision: 0.9678, Recall: 0.8771, F1 Score: 0.9202, Specificity: 0.8572
Fold 5
Fold 5 Metrics:
Accuracy: 0.8716, Precision: 0.9666, Recall: 0.8752, F1 Score: 0.9187, Specificity: 0.8544

Cross-Validation Results (Pre-trained Model):
Average Accuracy: 0.8731
Average Precision: 0.9672
Average Recall: 0.8766
Average F1 Score: 0.9197
Average Specificity: 0.8562
```

Profit

The model's average profit per business is $6,416.44. If applied to 1,000 businesses, this would generate $6,416,440. The total profit from the validation set with 174,547 rows amounts to $1,119,969,849.60.

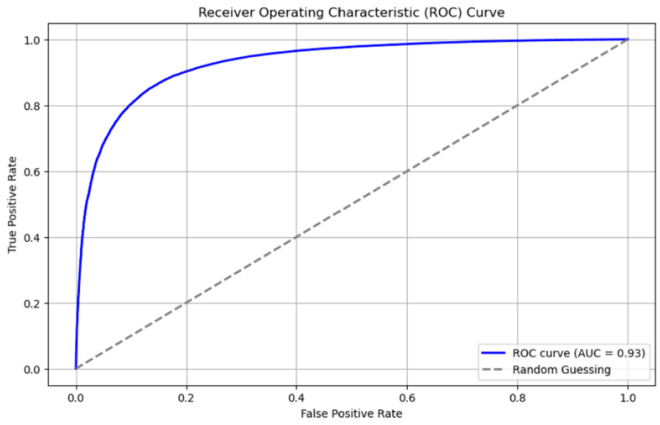
```
Total Net Profit/Loss: 1119969849.60
Average Total Net Profit/Loss: 6416.44
```

Below, the normal distribution chart shows that most profit/loss values are clustered around zero, with a rightward skew indicating fewer instances of higher profits. On the left side, there are very few negative values (losses), suggesting that the model is effective at minimizing losses.



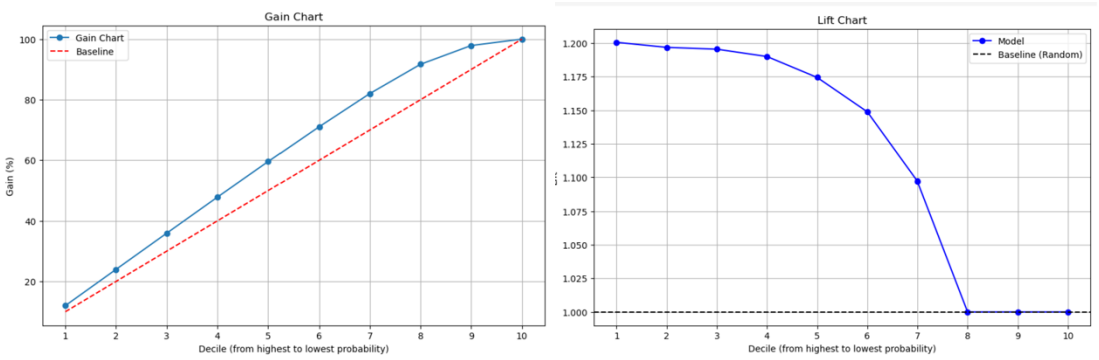
AUC-ROC

Our neural network's AUC of 0.93 indicates excellent performance, as it effectively distinguishes between businesses that will pay the loan in full and those that will default. This high AUC demonstrates the model's reliability and accuracy in making loan repayment predictions.



Gain & Lift Chart

Below is our gain chart, which reflects the model's performance in prioritizing the correct classification. While the gains chart shows suboptimal results, it's important to note that neural networks are typically not optimized for generating probability rankings, which is why their performance in this type of evaluation can be less reliable. For more accurate evaluation, metrics like the ROC curve are more appropriate for neural networks as they better reflect the model's ability to distinguish between classes.



The lift chart above shows a subpar performance in the lower deciles is not concerning because MLPs are designed for classification, not ranking. The model performs well in the top deciles, where decisions are most impactful. Lift charts, sensitive to calibration, don't fully reflect the model's classification power, and class imbalance naturally affects lower-decile performance, which is expected and not a failure. Real-world applications typically focus on higher deciles, where the model excels.

5.5 Classification Trees

In this section, we will explore classification trees, single bagging, and boosting techniques. Classification trees help in making predictions by recursively splitting the data into subsets based

on feature values, creating a model that is easy to interpret. Single bagging improves model stability by combining multiple bootstrap samples, reducing variance and overfitting. Boosting, on the other hand, builds a series of weak learners sequentially, with each new model correcting the errors of its predecessor, resulting in improved model performance. These techniques will be analyzed to compare their effectiveness in handling the dataset and improving classification accuracy.

Feature Engineering

We dropped columns that were likely adding unnecessary noise to the dataset. To assess this, we used frequency visualizations to observe the occurrence of values in each column. By removing the columns NAICS_U.S._Industry_Title, Business_Type, and Urban_Rural_Status, we aimed to test whether their removal improves the performance of tree-based models. This step was intended to reduce noise and prevent overfitting by eliminating potentially redundant or non-informative features. Additionally, this reduction in dimensionality helps streamline the model, allowing it to focus on the most relevant features for prediction.

Training & Validation

To get the best results, we experimented with different data splits for each tree model. This allowed us to fine-tune the amount of training and validation data, ensuring that each model was optimized for performance. By adjusting the splits based on the model's needs, we were able to enhance the accuracy and generalization of the predictions for each algorithm.

We did use dummy variables for categorical features, such as NAICS_U.S._Industry_Title, Business_Type, and Urban_Rural_Status. This step allowed the tree models to handle categorical data effectively, as trees require numerical inputs. By converting these categories into dummy variables, the models could evaluate them during the splitting process.

However, we didn't apply normalization or dimensionality reduction for the tree models, as these algorithms are not sensitive to feature scaling. Tree-based models can handle raw, unscaled features and identify important splits without the need for dimensionality reduction. This allows them to work effectively with high-dimensional data.

Hyperparameters

Single Tree (Decision Tree Classifier)

- **Max Depth = 20:** Limiting the depth helps prevent overfitting by ensuring the tree doesn't grow too complex and capture noise in the data. A depth greater than 20 would risk overfitting, making the model overly sensitive to small variations in the training data.

- **Max Leaf Nodes = 150:** This controls the complexity of the model, limiting the number of leaves to 150 to avoid overfitting while maintaining enough flexibility to capture essential patterns in the data.
- **Class Weight = 'balanced':** Given the class imbalance in the target variable, this parameter ensures that the model treats both classes (charged-off and paid in full) with equal importance, reducing bias toward the majority class and improving the overall model accuracy.

Bagging (Random Forest)

- **DecisionTreeClassifier:** The decision tree used as the base estimator was configured with the same hyperparameters as the single tree model to ensure consistency and prevent overfitting.
- **n_estimators = 50:** This hyperparameter sets the number of trees in the random forest ensemble, helping to improve model robustness by averaging predictions across multiple trees, leading to better generalization.
- **bootstrap = True:** Enables bootstrapping, where each tree in the ensemble is trained on a random subset of the data, promoting diversity among the trees and reducing overfitting.
- **max_samples = 0.8:** Limits each tree to a random 80% of the data, helping to reduce overfitting by ensuring that each tree is trained on a slightly different sample of the data.
- **max_features = 0.7:** Randomly selects a subset of features for each tree, enhancing diversity and ensuring that individual trees do not become overly dependent on specific features.
- **oob_score = True:** Utilizes out-of-bag samples (data points not used for training a specific tree) to estimate model performance, allowing for validation without the need for a separate validation set.

Boosting (XGBoost)

- **learning_rate = 0.2:** Controls the step size at each iteration of the boosting process. A value of 0.2 strikes a good balance between fast learning and convergence without overshooting the optimal solution.
- **min_child_weight = 15:** Specifies the minimum sum of the instance weight needed in a child node. A higher value helps avoid overfitting by ensuring that each split has enough data to be meaningful.
- **n_estimators = 300:** Sets the number of boosting rounds or trees. A value of 300 ensures enough iterations to learn complex patterns, while also preventing premature convergence.
- **scale_pos_weight = sum(y_train == 0) / sum(y_train == 1):** This parameter adjusts the weights of positive and negative classes in the imbalanced dataset, helping the model focus more on predicting the minority class (defaults) without bias.

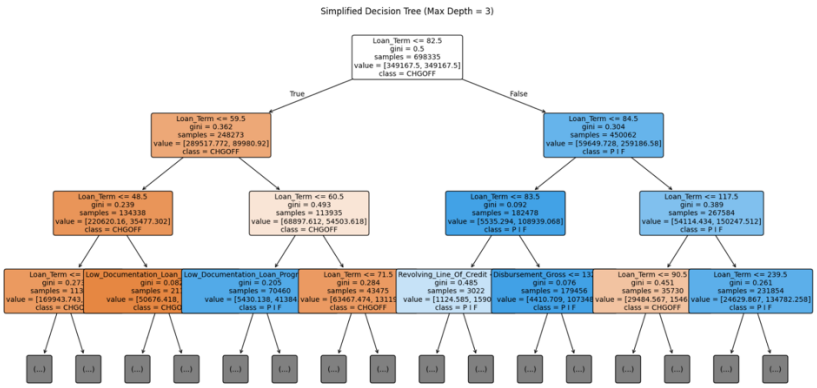
Metrics

Single Tree (Decision Tree Classifier)

The single tree model achieved metrics above 0.9, indicating strong performance in distinguishing between classes. This suggests high accuracy and reliability in predictions without significant overfitting.

Training Set Metrics:	Test Set Metrics:
Accuracy: 0.9057	Accuracy: 0.9042
Precision: 0.9785	Precision: 0.9781
Recall: 0.9062	Recall: 0.9047
F1 Score: 0.9409	F1 Score: 0.9400
Specificity: 0.9034	Specificity: 0.9016

Below is the single decision tree with a maximum depth of 3. This depth helps prevent overfitting by simplifying the model while still capturing key patterns for classification. It clearly shows how the model makes decisions based on the selected features.



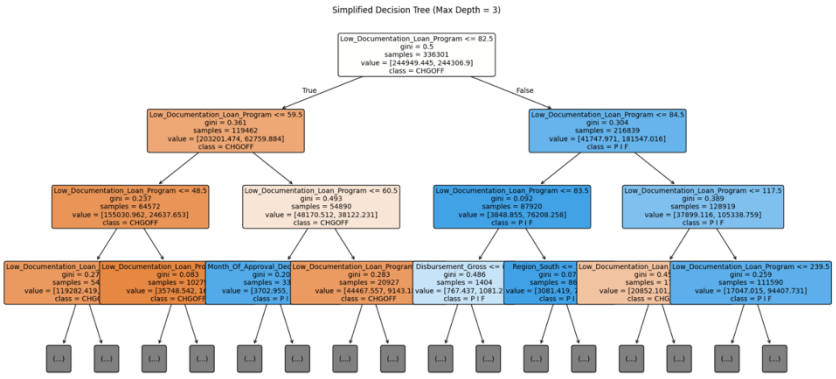
Bagging (Random Forest)

The Bagging model also achieved metrics above 0.9, indicating strong performance in classifying loans accurately. By using multiple trees and introducing randomness with bootstrapping, Bagging reduced overfitting and improved generalization. The model's ability to average over several trees contributed to its robust predictive power.

Training Set Metrics:	Test Set Metrics:
Accuracy: 0.9066	Accuracy: 0.9054
Precision: 0.9792	Precision: 0.9793
Recall: 0.9066	Recall: 0.9051
F1 Score: 0.9415	F1 Score: 0.9408
Specificity: 0.9068	Specificity: 0.9069

Below is the Random Forest Bagging model, which uses multiple decision trees with bootstrapped data samples. By averaging over several trees and introducing randomness, this model reduces overfitting and improves generalization. Each tree contributes to the final

prediction, allowing the model to capture more complex patterns while avoiding the risk of overfitting that a single tree might face.



Boosting (XGBoost)

XGBoost achieved high metrics, with most above 0.9, but some, like accuracy and recall, fell slightly below. We used a learning rate of 0.2 to speed up convergence and improve performance, focusing on optimizing the learning rate without adding unnecessary complexity.

Training Set Metrics:	Test Set Metrics:
Accuracy: 0.9010	Accuracy: 0.8966
Precision: 0.9795	Precision: 0.9776
Recall: 0.8994	Recall: 0.8958
F1 Score: 0.9377	F1 Score: 0.9349
Specificity: 0.9088	Specificity: 0.9004

Cross Validation Folds

Since random forest was our best-performing model, we only performed cross-validation on it. The results show no signs of overfitting, indicating that the model generalizes well and maintains consistent performance across different subsets of the data.

```
Metrics for each fold:
Fold 1 - Precision: 0.9787, Recall: 0.9060, F1 Score: 0.9409, Accuracy: 0.9057, Specificity: 0.9046
Fold 2 - Precision: 0.9785, Recall: 0.9078, F1 Score: 0.9418, Accuracy: 0.9070, Specificity: 0.9035
Fold 3 - Precision: 0.9787, Recall: 0.9078, F1 Score: 0.9419, Accuracy: 0.9072, Specificity: 0.9044
Fold 4 - Precision: 0.9788, Recall: 0.9056, F1 Score: 0.9408, Accuracy: 0.9056, Specificity: 0.9052
Fold 5 - Precision: 0.9790, Recall: 0.9073, F1 Score: 0.9418, Accuracy: 0.9070, Specificity: 0.9057

Average Metrics across all folds:
Average Precision: 0.9788
Average Recall: 0.9069
Average F1 Score: 0.9415
Average Accuracy: 0.9065
Average Specificity: 0.9047
```

Profit

Single Tree (Decision Tree Classifier)

For profit, the single decision tree achieved the second-highest result with $7,406.42. This demonstrates the model's ability to predict profitable loans while maintaining strong overall performance.

```
Total Net Profit/Loss: 1293094098.05
Average Total Net Profit/Loss: 7406.72
```

Bagging (Random Forest)

The random forest model achieved the highest profit with $7,452.13, outperforming both the single decision tree and boosting. This demonstrates its superior ability to capture complex patterns and improve profitability.

```
Total Net Profit/Loss: 1951532847.00
Average Total Net Profit/Loss: 7452.13
```

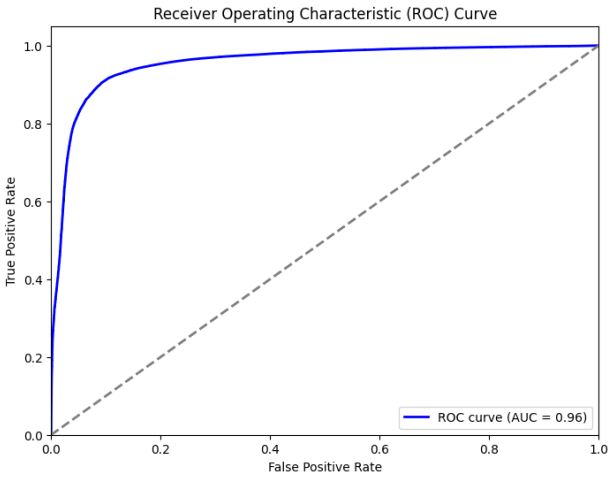
Boosting (XGBoost)

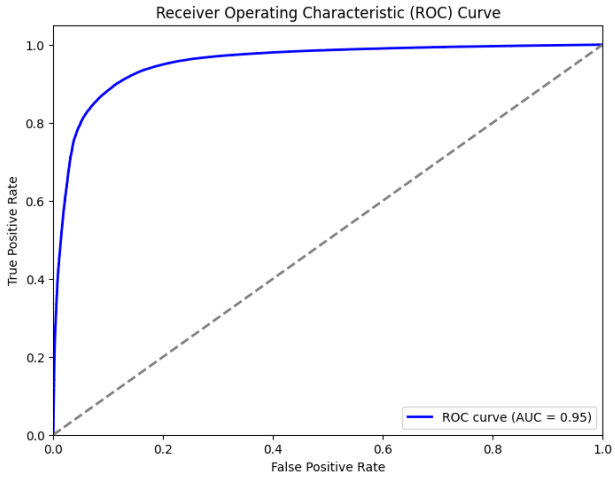
Boosting (XGBoost) performed the worst out of the three models with a profit of $7,123.60, but still showed strong results. While it didn't outperform the random forest, it still demonstrated the model's ability to make valuable predictions.

```
Total Net Profit/Loss: 2487332690.10
Average Total Net Profit/Loss: 7123.60
```

AUC-ROC

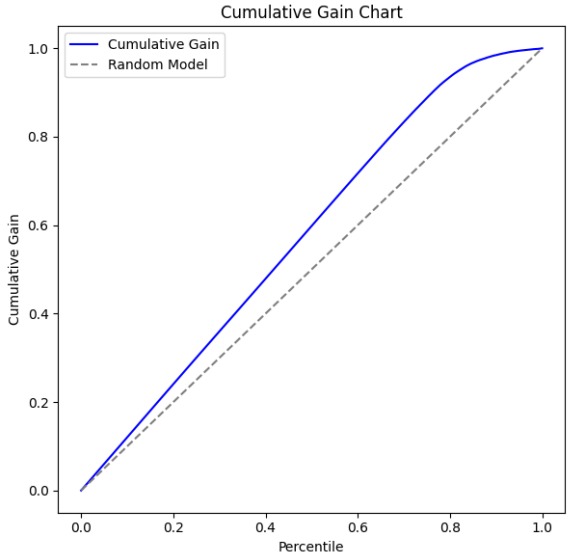
Random forest achieved an AUC-ROC of 0.96, outperforming both the single decision tree and boosting models, which each had an AUC-ROC of 0.95. This indicates that the random forest model has a slightly better ability to distinguish between the two classes.

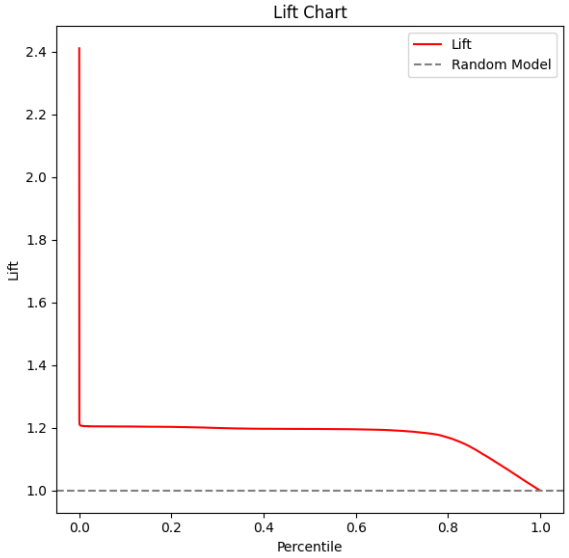




Lift & Gain Charts

Below are the lift and gain charts for the random forest model, which show that it outperforms random chance and demonstrates a relatively low lift. This suggests that the model is effective in identifying high-value predictions while minimizing unnecessary overfitting.





6 Optimal Model Selection

In the context of the bank's loan default prediction, Random Forest (bagging) offers several advantages over individual decision trees. By aggregating multiple decision trees, Random Forest improves predictive accuracy and robustness, reducing the likelihood of overfitting. Each tree in the forest is trained on a random subset of the data, and the final prediction is made by averaging the results from all trees. This helps ensure that the model isn't overly sensitive to individual data points or noise, which is especially important in the financial sector where data can be noisy and complex. Random Forest's ensemble approach allows it to capture a wide range of relationships in the data, including interactions between features, making it highly effective for predicting loan defaults.

Moreover, Random Forest's ability to handle both numerical and categorical variables without the need for extensive preprocessing is another advantage. It automatically selects the most important features for decision-making, and the model's performance remains strong even when some features are irrelevant or noisy. This feature is especially useful in the bank's scenario, where many factors, such as business type or credit history, could be relevant but also highly variable. By identifying the most important factors that contribute to loan defaults, Random Forest helps the bank make more informed, data-driven decisions. Its inherent flexibility, combined with its ability to produce more reliable and stable predictions compared to single decision trees, makes it a powerful tool for the bank's risk assessment needs.

7 Implementing the Model for Optimal Results

We have provided a video demonstration on how to use the model effectively. In summary, the process begins by uploading your data into a data cleaning notebook, where the dataset is preprocessed and prepared for model input. After cleaning and transforming the data, the model processes it to predict whether the bank should approve or reject a loan application. The final output includes both the decision—'Approve' or 'Reject'—as well as the probability associated with each decision, providing valuable insights for loan approval processes.

For a step-by-step guide, refer to the video: Loan Approval Prediction Model Demonstration

8 Data Analysis & Business Strategies for Banks

.

We have prepared a set of slides to guide you through the data analysis and business strategies for banks. These slides provide a comprehensive overview of how the model can be applied to real-world banking scenarios. They cover key aspects such as analyzing customer data, identifying trends, and making data-driven decisions to optimize loan approval processes. Additionally, the slides highlight how business strategies can be enhanced using the insights gained from the model's predictions.

For a detailed presentation, please refer to the slides: Loan Approval Business Strategy Deck

9 Limitations and Future Work

9.1 Limitations

Outdated Data: The dataset used in the model was not as recent, which could impact its relevance for current loans. Since financial conditions, lending policies, and economic factors can change over time, using outdated data may affect the model's accuracy in predicting the approval of loans in the current environment.

Class Imbalance: The dataset had more data for defaulted loans than for loans that were paid in full. This imbalance can affect the model's performance by making it biased toward predicting defaults, potentially leading to suboptimal predictions for non-defaulting loans. While techniques like balancing class weights were implemented, the overrepresentation of defaulted loans may still influence the model's ability to predict loans that are likely to be paid in full, especially in real-world scenarios where the distribution of outcomes may be more balanced.

9.2 Future Work

Obtaining New Data: One of the most important next steps is to gather more recent and up-to-date data. This will help ensure that the model remains relevant to current financial trends, lending policies, and economic conditions. By using more current data, we can enhance the model's predictive accuracy and ensure that the bank's decisions are based on the latest available information.

Incorporating Additional Features: Adding more features to the dataset could improve the model's performance by capturing additional relevant factors that influence loan approvals. Potential features could include credit scores, income levels, transaction history, or even external economic indicators. This expansion would allow the model to make more nuanced predictions, further increasing its reliability and value.

Deployment to a Bank: Another exciting opportunity for future work is to deploy the model to an actual bank. By implementing the model in a real-world environment, we can assess its performance under live conditions, gather feedback, and make further refinements. This would allow us to directly measure its impact on the bank's loan approval process, improving efficiency and decision-making.

10 Conclusion

In this project, we built a predictive model to assist banks in determining whether to approve or reject loans. The goal was to improve decision-making by accurately identifying which businesses are likely to repay their loans in full and which may default. To achieve this, we explored multiple machine learning techniques, including classification trees, neural networks, logistic regression, discriminant analysis, and KNN. Each method was tuned and tested for optimal performance, with a focus on minimizing overfitting and handling the class imbalance between defaulting and non-defaulting businesses.

The random forest model outperformed other approaches, achieving the highest average net profit and AUC-ROC score. It effectively captured complex relationships in the data while maintaining interpretability and generalizability. Single decision trees, although effective, had slightly lower performance metrics, while boosting performed well but lagged behind random forest in terms of profitability. We used cross-validation on the random forest model to ensure its robustness and validate that it was not overfitting, confirming its reliability for real-world deployment.

Data preprocessing played a critical role in the success of the models. By removing unnecessary columns, handling categorical variables with dummy encoding, and carefully splitting the data, we were able to optimize the data for the models. Furthermore, hyperparameter tuning was performed to refine model performance, ensuring that each algorithm functioned at its best. Our attention to detail in these areas directly contributed to the strong performance of the final models.

Looking ahead, there is significant potential for improvement. The inclusion of more recent and relevant data, the addition of more features, and deployment of the model in a live banking environment could further enhance its accuracy and practical utility. By continually refining the model, incorporating user feedback, and expanding its scope, we can provide banks with a powerful tool for making more informed loan decisions, ultimately improving profitability and reducing financial risk.