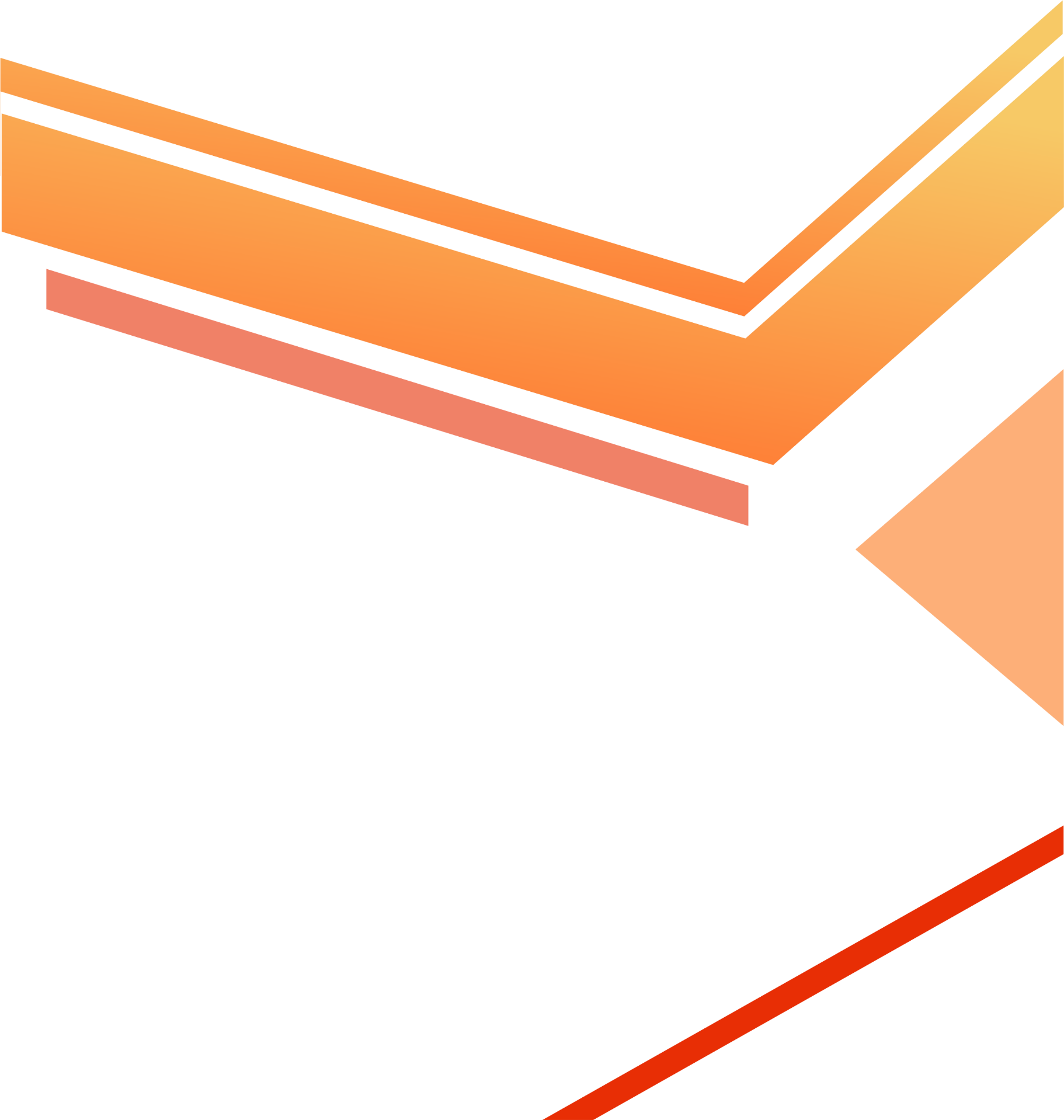


**BANA 273 Final Report – Dream Housing Finance Loan Prediction**

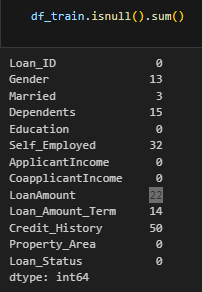
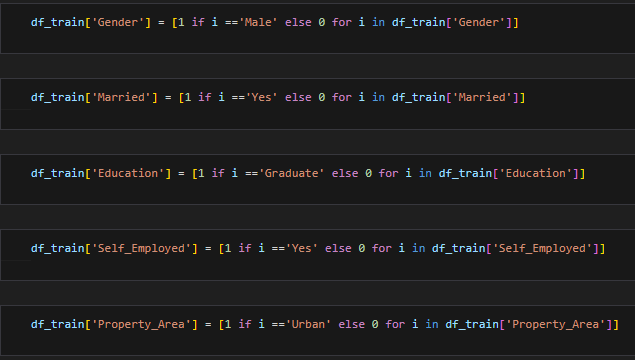
By Group 5B – Abby Chung, Ethan Le,

Sai Kulkarni, Yangky Tan



When in search of the right housing, we wish to find accommodation fast, but as painlessly as possible. Being granted the right loan is always an important step. In modern society, due to the rapid advancement of technology, many job functions are being automated, and for Dream Housing Finance, assessing loan eligibility is not an exception. For individuals to apply for a housing loan, a variety of factors that must be considered include credit score, income, marital status, assets, and job history. If a predictive model could be generated that would automate the assessment of all these factors and mark applicants for approval or rejection quickly, it would significantly reduce the time and effort of both the employees assessing the loan and the candidate applying for it.

We have obtained our dataset on Kaggle from the Dream Housing Finance Home Loan Dataset that consists of customer data. The business problem we are trying to solve is the creation of a predictive model that can accurately classify a potential customer as one who will be approved for a loan or one who will not be approved for a loan. If the model is strong, we can use it to automate the loan approval process, saving valuable time and resources for the Dream Housing Finance company.

**Figure 1. Null values Column**  **Figure 2. Value Change in Binary Column**

**3. Data Summary**

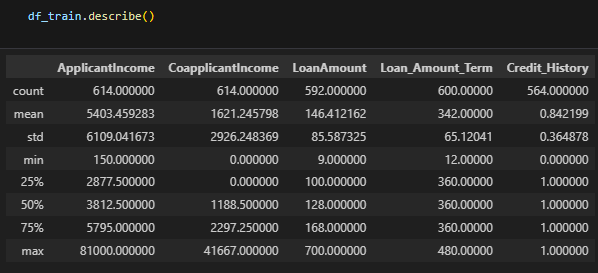
**(a) Features & Classes**

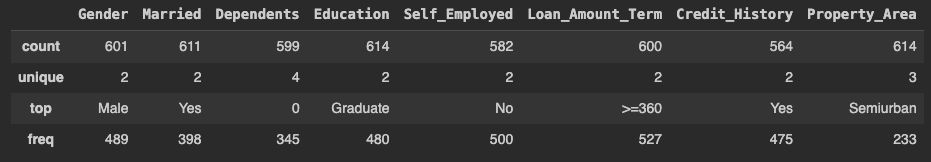
Looking at the data, there are 615 rows and 13 columns, which leads to 165 x 13 = 7995 expected datapoints. However, we have 149 nulls, which gives us about 1.8% of our expected data missing as stated in Figure 1. We had 13 columns, so this translates to 12 features and 1 classification column, which are as follows:

* Loan\_ID – the identification number associated with the customer requesting a loan
* Gender – The customer’s gender, possible values were M/F
* Married – The customer’s marital status, possible values were Y/N
* Dependents – How many dependents the customer has, possible values were 0,1,2,3+. Note that this data was inputted categorically, not numerically.
* Education – Whether the customer was a college graduate or not. Possible values were Graduate/Not Graduate
* ApplicantIncome – Applicant Income. This data was continuous.
* CoapplicantIncome – Income of a person who is applying alongside with the applicant
* LoanAmount – Amount applied to be granted as loan (in thousands)
* Loan\_Amount\_Term – The term of the loan (in months)
* Credit\_History – Whether the customer’s credit history meets guidelines. Possible values were Y/N
* Property\_Area – Where the customer lives. Possible values were Rural/Suburban/Urban
* Loan\_Status – Whether the customer qualified for a loan. Possible values were Y/N. **This was our class.**

In order to create a more accurate predictive model, clean data is required. We took note of two ways of dealing with nulls - dropping them from the dataframe completely or filling in the values. Additionally, some predictive models need data to be numeric in nature, and so in various columns we encoded categorical data into numerical data. For instance, in the gender section, we define female as 0 and male as 1 as illustrated in Figure 2. Afterward, we removed the Load ID column since it was an identifying column for customers, which is not relevant to our analysis.  These were necessities that we noticed at an initial glance, but we wanted to explore the data more before deciding on how we would process the data.

**(b) Data Exploration**

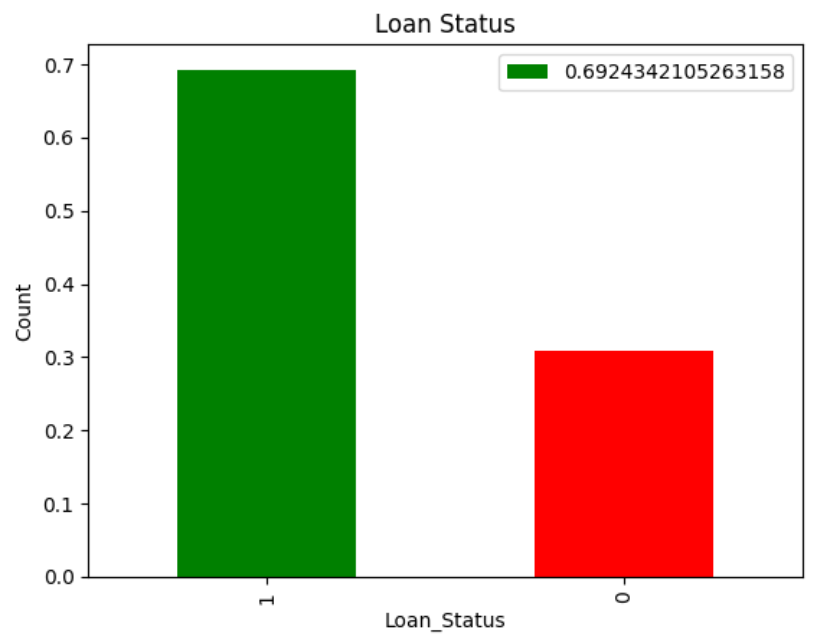




**Figure 3. Descriptive Statistical Value Table**

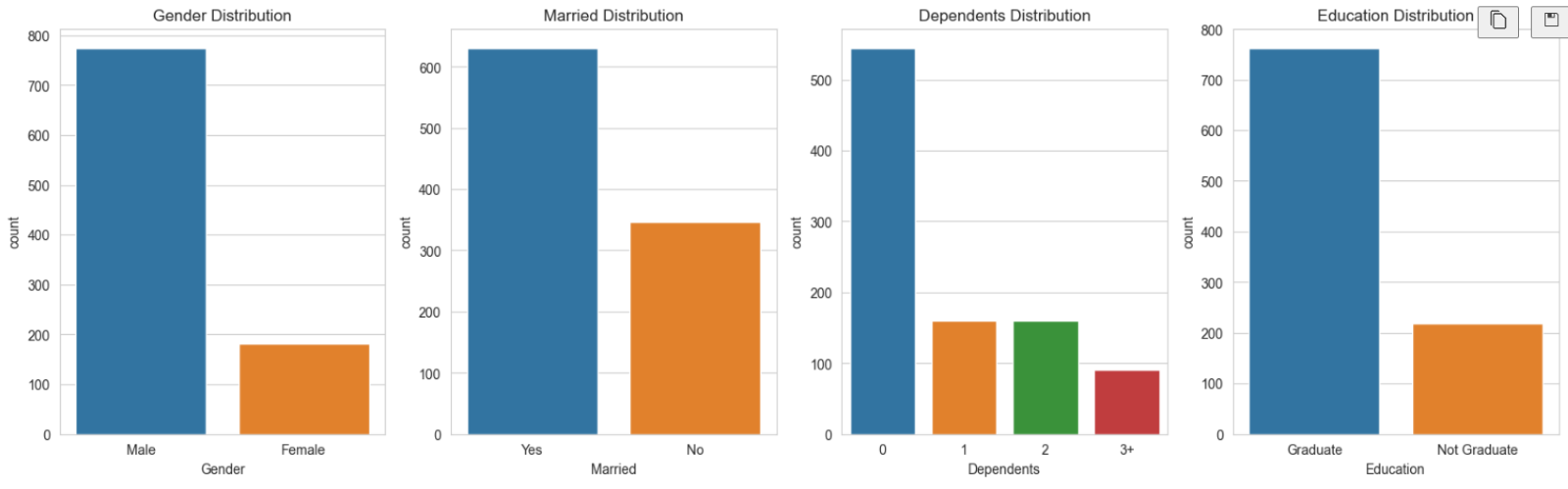
After observing the features and classes in the dataset, we began data exploration. First, we applied the describe function to obtain the descriptive statistical values on the columns containing numeric values as pointed out in Figure 3. From there, we engaged in both univariate and bivariate analyses through visualizations. The methodology of both methods was very similar, and involved bar plots, boxplots, and histograms. However, univariate analysis led us to see the distribution and characteristics for each feature, while bivariate analysis was used to determine the relationship between a feature and the class. We will not be speaking on the results for every feature, but rather we will go over the ones we found to be most important.

Before looking at the features in the data, it was important to understand the class distribution in our dataset. We counted the normalized values for the classes and visualized it, coming up with the results pictured in Figure 4.



**Figure 4 – Distribution of Class (Loan Approval Status)**

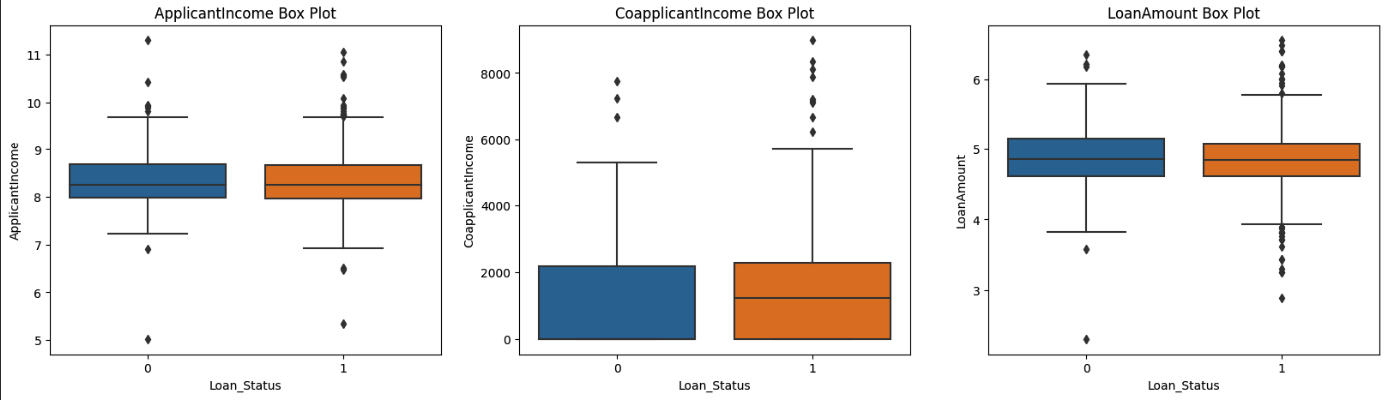
As we can see, the distribution of classes is 69.24% approved, so we want to beat the accuracy of 69.24% with our models, as that’s how a model that simply approved all applicants would perform. After figuring out our class distribution, we continued with our univariate analysis to determine the quality of our data. As can be seen in Figures 5A and 5B, the data seemed heavily skewed in every feature except for property\_area, meaning that customers were highly likely to have certain characteristics, such as being male, being married, being a college graduate, and having good credit history to name a few.



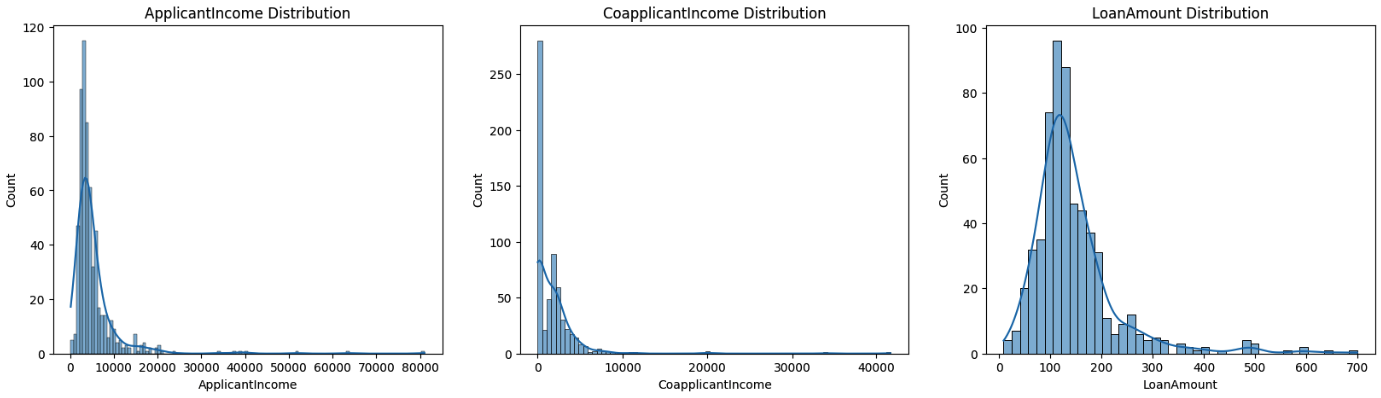


**Figures 5A and 5B – Feature Distributions**

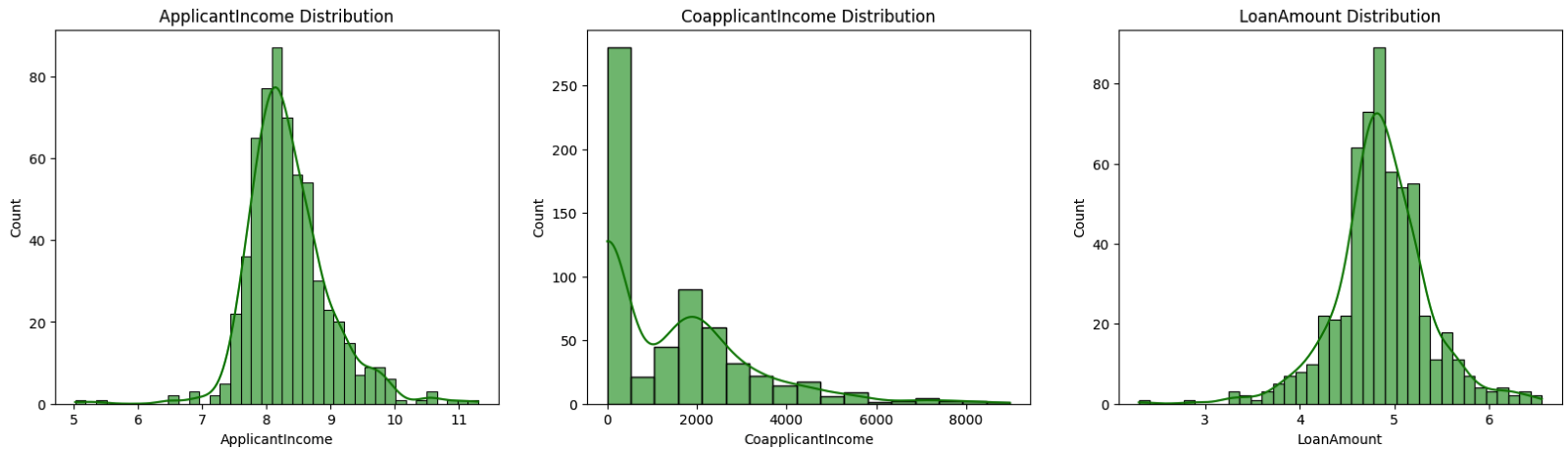
Continuing with our univariate analysis, we wanted to check for outliers before moving onto bivariate analysis, and we identified outliers using histograms and boxplots, which can be observed in Figures 6A and 6B. Through this analysis, we found that there were indeed numerous outliers in the Coapplicant Income and Applicant Income columns. Notably, when beginning our bivariate analysis, we found many high-income outliers among those who had good credit history, and fewer among those with bad credit history, as can be seen by our boxplots and histograms pictured in Figures 6A and 6B. We would control for this, and the resulting distribution of data can be seen in Figure 6C, but we will discuss methodology later in the data preprocessing section.



**Figure 6A - Boxplots plotting ApplicantIncome, Coapplicant Income, and LoanAmount against the class. Many outliers can be observed**

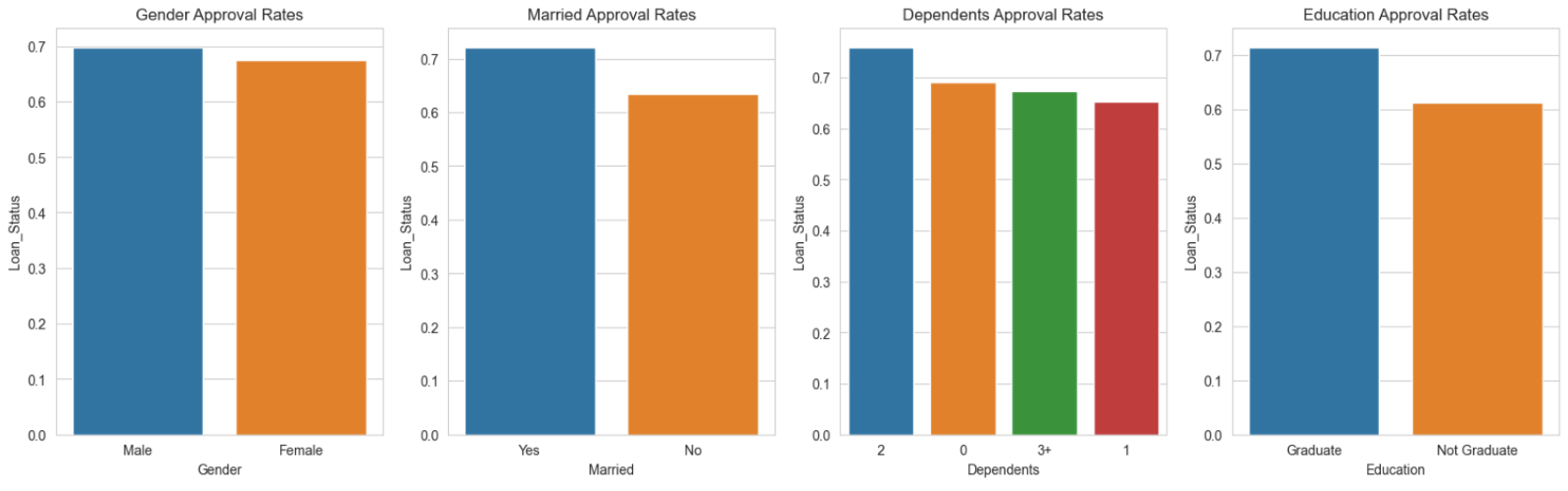


**Figure 6B - Histograms showcasing the distribution of datapoints from the columns observed in Figure 6A**



**Figure 6C - Histograms from Figure 6B after methods to control outliers were applied, creating data that is better for modelling.**

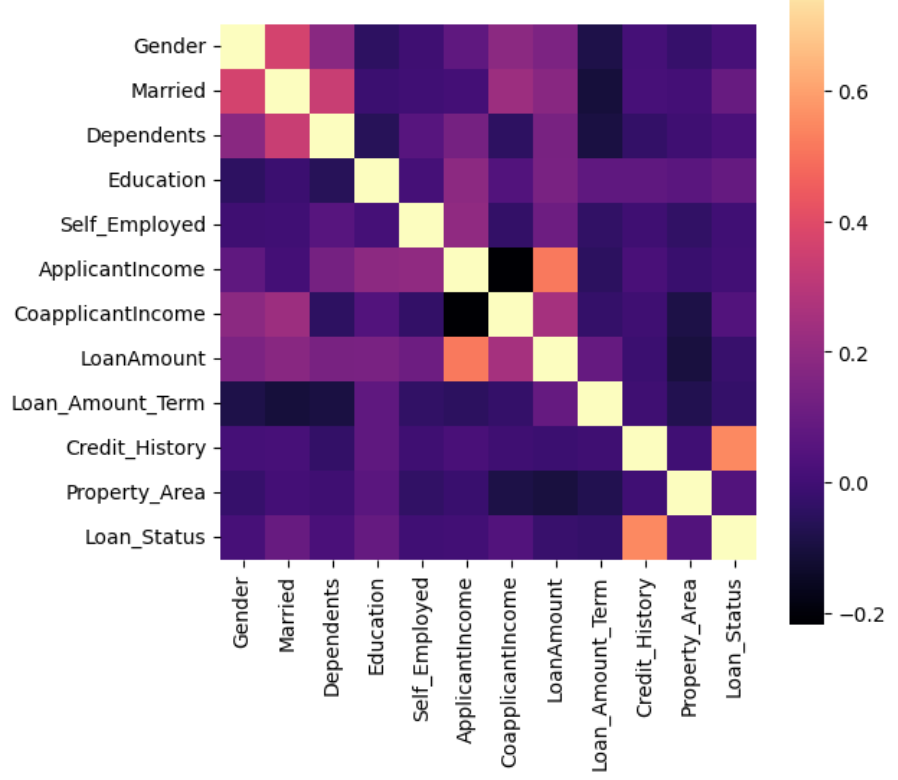
The boxplot started our bivariate analysis, and continuing with it, we visualized each feature and its class distribution, looking for relationships between features and classification to figure out which features were important. Our visualizations can be seen in Figures 7A and 7B, and the insights we obtained from it are that very few features in our dataset have a strong impact on classification. In fact, the only one with obvious correlation was Credit History.





**Figures 7A and 7B – Bivariate Analysis, Plotting Features vs Loan Approval Status**

We confirmed these findings through the use of a correlation matrix (Figure 8), and found indeed that that the only feature with a large effect on Loan Status was Credit History. Interestingly, we did find a negative correlation between Applicant Income and Coapplicant Income, which makes sense as it is easier to get a loan with a weak credit score if you have a co-applicant who has a strong credit score.



**Figure 8 – Correlation Matrix Examining Feature Correlations**

**4. Analysis**

**(a) Models Used and Justification for Use**

The models we used were Logistic Regression and Decision Trees, and we used various ensemble models based on decision trees – namely Random Forest, ADABoost, and XGBoost. Our reasoning behind trying logistic regression was that our dataset was relatively small, and logistic regression models are known to perform well on small datasets. Decision trees control for feature importance on their own, and though we had an idea as to the importance of each feature, we hoped that decision trees would help us confirm this. We used two simple models to start, so we wanted to add some complexity to our analysis and see if we could produce more robust models by using the ensemble models.

**(b) Benchmarks**

To obtain our benchmarks, we ran our models on the data without any preprocessing, except for dropping null values as some models were unable to function with nulls still in the data. As a reminder, the base accuracy we wanted to beat was 69.24%, as this is how a model would perform if it were to just classify all cases as approved for a loan. Additionally, we did not modify any hyperparameters for this test. However, for the sake of consistency, we did 10-fold cross-validate the models, as we did so in further tests and wanted to truly understand whether models were performing better or worse with different preprocessing methods and hyperparameter tuning. Our results were as follows:

(i) Logistic Regression: Mean accuracy of 0.799

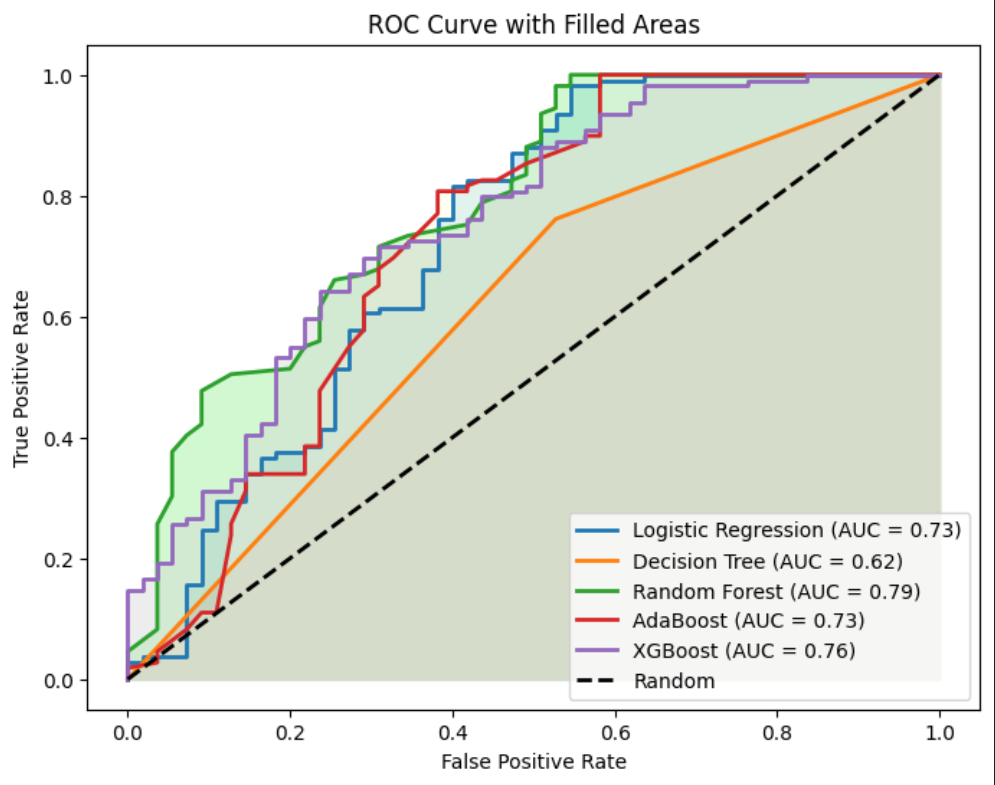
(ii) Decision Tree: Mean accuracy of 0.7146

(iii) Random Forest: Mean accuracy of 0.7958

(iv) ADABoost: Mean accuracy of 0.7918

(v) XGBoost:  Mean accuracy of 0.7833

Overall, the mean accuracy for the benchmarks were very close, with Decision Tree being notably weaker without any hyperparameter tuning nor pruning. Now that we had our benchmarks, we were ready to begin preprocessing data and tuning hyperparameters to improve our models and ultimately select the best one for identifying whether one should be approved for a loan. We plotted the ROC/AUC curves as well, confirming that for the most part Random Forest was performing the best on the benchmark.



**Figure 9 – ROC/AUC Curve Plot: Random Forest Performs Best**

**(c) Preprocessing methods used**

There were several different steps we took to attempt to improve our model accuracy through the preprocessing of data. We dropped the nulls in our benchmark, so we tried replacing them with the median and mean for numerical data, or mode for categorical data. We also tried to implement binning to simplify the data and control for outliers. As an alternative method of controlling for outliers, we combined a log transformation on the Applicant Income and Loan Amount with a threshold based on Coapplicant Income. Our data had close to a 70/30 split on its classification distribution, and our models were underperforming on cases where the true class was a rejected loan, so we tried including more of this class with 80/20, 60/40, and 50/50 splits through proportional sampling. Lastly, we combined various preprocessing methods and tried them together, such as controlling for outliers while replacing nulls, and doing that while also proportionally sampling or binning.

**(d) Results for each model**

We have included tables with our experiment results, as well as an exploration of the results for each model.

**(i) Logistic Regression:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | **GridSearch CV best score :** | **GridSearch score test** | **Accuracy score** | **Solver** | **Penalty** |  |
| 1 | 0.8279 | 0.8056 | 0.8055 | liblinear | l1(c-value: 100) | 80, 20 (TRAIN, TEST SPLIT) |
| 2 | 0.8169 | 0.8395 | 0.8395 | liblinear | l1(c-value: 100) | 70, 30 (TRAIN, TEST) |
| 3 | 0.6916 | 0.6975 | 0.8395 | sag | l1 | 70, 30(TRAIN, TEST) |
| 4 | 0.8169 | 0.8395 | 0.8395 | liblinear | l2(c value - 0.1) | 70, 30(TRAIN, TEST) |
| 5 | 0.8169 | 0.8395 | 0.8395 | newton-cg | l2(c value: 0.1) | 70, 30(TRAIN, TEST) |
| 6 | 0.707 | 0.6389 | 0.8055 | saga | l2(c-value: 100) | 70, 30(TRAIN, TEST) |

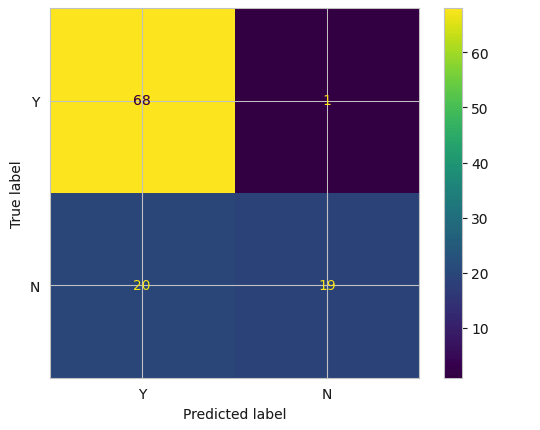
**Table 1 – Logistic Regression Experiment Results**

We performed logistic regression modeling on the given dataset, utilizing the liblinear solver and applying an L2 penalty. Through the implementation of grid search cross-validation with different cross-validation values, we derived results that are detailed in the first row of the table. However, the confusion matrix raised concerns as it exhibited less than satisfactory performance—specifically, the false negative values outnumbered the true negative values. This suggests a potential limitation in our model's ability to accurately identify negative instances.

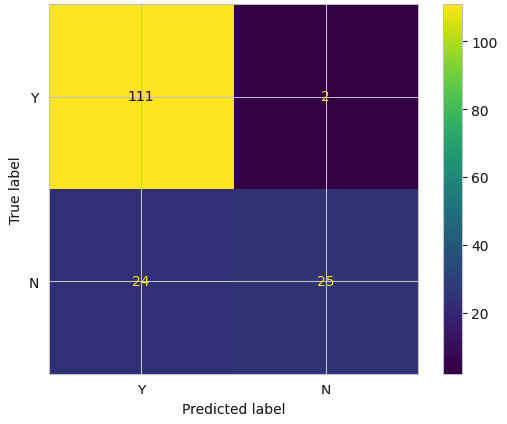
Logistic Regression has quite a few hyperparameters to work with – **penalty**, which specifies which type of regularization is used, **c**, which is the inverse of the strength of the regularization, **solver**, a hyperparameter that can be used to change the algorithm used for optimization, **max\_iter**, which determines the maximum number of iterations taken for the solver to converge (finish optimization),fit\_intercept, which determines whether an intercept is calculated or not, and class\_weight, which is used to handle class imbalance.

We conducted experiments by systematically varying solver, penalty, and C values during the logistic regression modeling process. Our exploration included trying different solvers such as newton-cg, sag, and saga. Through this experimentation, we observed that the choice of solver could significantly influence the model's performance.

 Additionally, we found that adjusting the train-test split ratio had an impact on the results. Notably, a split of 70:30 yielded better outcomes compared to a 80:20 split. This suggests that allocating a larger proportion to the training set enhanced the model's ability to generalize to new, unseen data. For co199mparison, we have Figures 10A and 10B, the confusion matrices for experiments 1 and 4 respectively.



**Figure 10A – Confusion Matrix for Experiment 1**



**Figure 10B – Confusion Matrix for Experiment 2**

Hence, for our dataset, we determined that employing the `liblinear` solver with a C-value of 0.1 and a 70:30 train-test split yielded the most favourable results in terms of accuracy and confusion matrix performance. However, it's important to note that the model continues to exhibit a notable issue with false negative values, indicating an area for further consideration and improvement. Therefore, exploring alternative models and conducting additional experiments is warranted to enhance overall predictive performance.

**(ii) Decision Tree:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Pre-processing** | **Type** | **Train Score** | **Test Score** | **Hyperparameters** |
| 1. | * All null values removed * Columns removed: Loan Id | DT | 1.00 | 0.716 |  |
| 2. | * All null values removed * Columns removed : Loan ID | DT - Pre Pruning | 0.825 | 0.775 |  |
| 3. | * All null values removed * Only the top 4 features considered | DT | 1 | .725 |  |
| 4. | * All null values removed * Only the top 4 features considered | DT - Pre Pruning | 0.833 | 0.794 |  |
| 5. | * All null values removed * Only the top 4 features considered | DT - Pre Pruning | 0.819444 | 0.7833 |  |
| 6. | * All null values removed * Only the top 4 features considered | DT - Post Pruning | 0.8777 | 0.7666 |  |
| 8. | * Null: Filled with mean/median * Only the top 4 features considered | DT | 1.0 | 0.72794 |  |
| 9. | * Null: Filled with mean/median * Only the top 4 features considered | DT - Pre Pruning | 0.83292 | 0.794117 |  |
| 10. | * Null: Filled with mean/median * Only the top 4 features considered | DT - Post Pruning | 0.828 | 0.7941 | Ccp-alpha : 0.01 |
| 12. | * Null: Filled with mean/median * Only the top 4 features considered * Binning (Applicantincome) | DT - Post Pruning | 0.8275 | 0.80147 | Ccp-alpha : 0.01 |
| 13. | * Null: Filled with mean/median * Only the top 4 features considered * Outliers treated | DT - Post Pruning | 0.8163 | 0.844 |  |
| **15..** | * Null: Filled with mean/median * Only the top 4 features considered * Outliers treated | Cross-validation |  |  |  |

**Table 2 – Decision Tree Experiment Results**

As indicated in Table 2, we conducted a series of 12 experiments with the decision tree model, incorporating various preprocessing techniques and hyperparameter tuning. Initially, we established a benchmark accuracy using minimal preprocessing. Subsequently, we explored different preprocessing steps in conjunction with hyperparameter tuning, incorporating both pre-pruning and post-pruning techniques through grid search cross-validation.

The hyperparameters used in our experiments were defined as follows:

```python

params = {'max\_depth': [2, 4, 6, 8, 10, 12],

          'min\_samples\_split': [2, 3, 4],

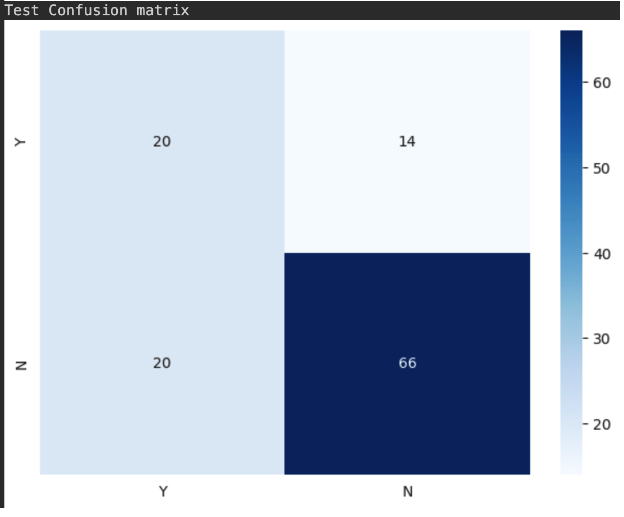
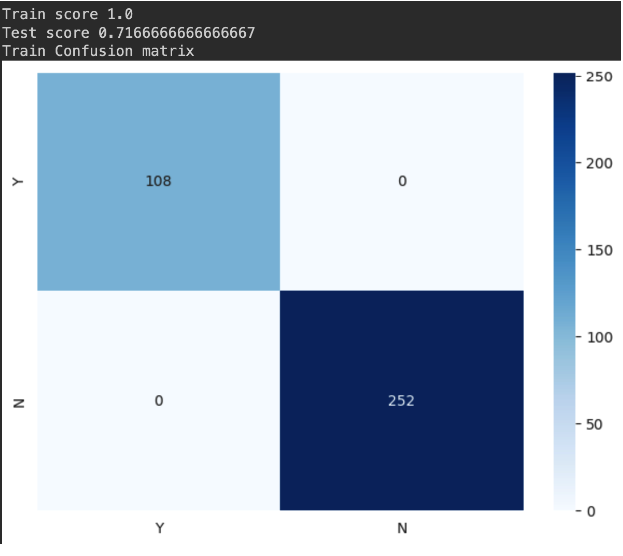
          'min\_samples\_leaf': [1, 2]}

```

The best-fit model, as determined through these experiments, was a DecisionTreeClassifier with parameters `max\_depth=2` and `min\_samples\_leaf=2`. For post-pruning, we further fine-tuned the model by adjusting the `ccp\_alpha` parameter, identifying the optimal value as 0.008.

Both pre-pruning and post-pruning procedures significantly enhanced the accuracy score and improved the distribution of the confusion matrix. Notably, these techniques effectively addressed the issue of overfitting in our decision tree model.

For the first experiment in table 1, the following confusion matrices were produced:

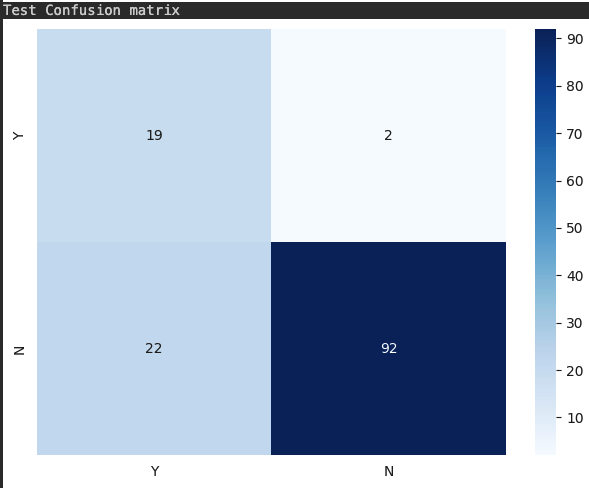
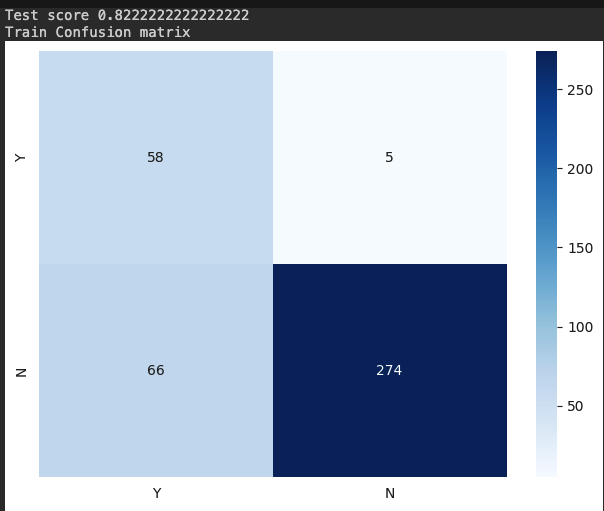
**Figures 11A and 11B – Decision Tree Experiment 1 Train and Test Confusion Matrices**

While the confusion matrix for the training data evidently reflects high accuracy and F1 score, the test data does not, indicating a prevalent overfitting issue. Furthermore, the disproportionate count of false positives raises concern. Consequently, we opted to address this challenge through pre-pruning, post-pruning, and grid search cross-validation techniques.

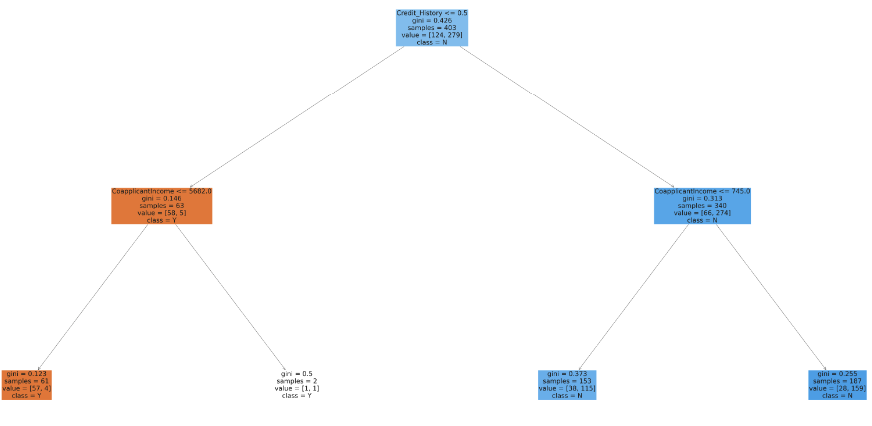
Optimal outcomes were achieved by employing pre-pruning, post-pruning, and grid search cross-validation for hyperparameter tuning, effectively mitigating the risk of overfitting.

The provided report includes the confusion matrix and decision tree obtained from the best results of pre-pruning and post-pruning. To enhance the presentation of comprehensive results, a separate document has been attached, encompassing detailed findings and outcomes.

* Pre-pruning confusion matrix and decision tree:

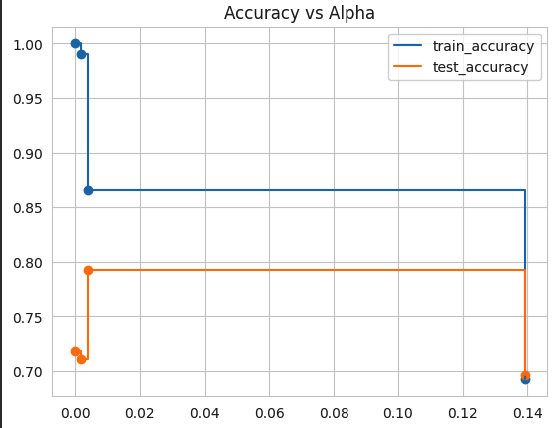


**Figures 12A and 12B – Decision Tree + Pre-pruning Train and Test Confusion Matrices**

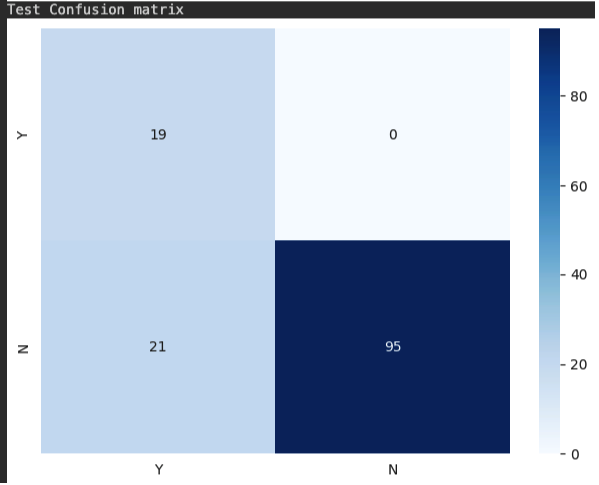
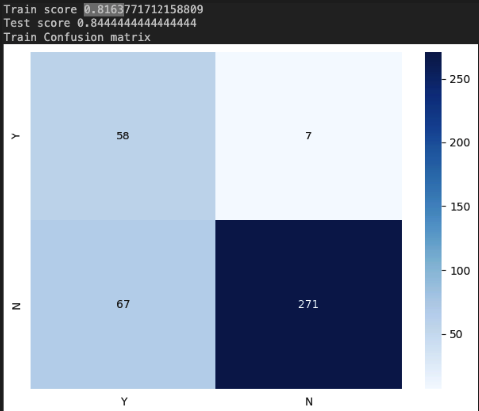


**Figure 13 – Best Pre-pruned Decision Tree**

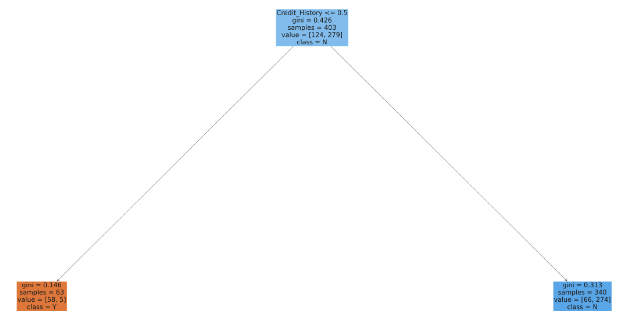
* Post-pruning ccp-alpha analysis, confusion matrix and decision tree:



**Figure 14 – Analysis of ideal value for ccp-alpha hyperparameter (post-pruning)**



**Figures 15A and 15B - Decision Tree + Post-pruning Train and Test Confusion Matrices**



**Figure 16 – Post-pruned Decision Tree**

As evident from the above confusion matrix, the overfitting issue has been successfully addressed, and the counts of true positive and true negative values are maximized. The decision tree, crucially splitting on the 'credit history' feature, demonstrates a discerning score for each division, with the number of samples ranging from 63 to 340. This indicates a more balanced and accurate representation of the model's predictive capabilities.

Hence, by implementing preprocessing steps that involve replacing null values with mean or median for numerical and categorical variables, addressing outliers, and focusing solely on top features, coupled with post-pruning techniques, we achieve the most optimal predictions for this dataset. The approach results in maximum accuracy and an improved confusion matrix, indicating the effectiveness of the applied strategies in enhancing predictive performance.

**(iii) Random Forest:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Pre-processing** | **Type** | **Model Accuracy** | **Hyperparameters** |
| **1** | * All null values removed * Only the top 4 features considered | Random forest | 0.7500 | N-estimators - 100  Random\_state = 0 |
| **2** | * Null: Filled with mean/median * Only the top 4 features considered | Random forest | 0.8015 | N-estimators - 40  Random\_state = 0 |
| **3** | * Null: Filled with mean/median * Only the top 4 features considered | Random forest | 0.8074 | N-estimators - 10  Random\_state = 0 |
| **4** | * Null:  Filled with mean/median * Only the top 4 features considered * Outliers treated | Random Forest | 0.8148 | N- estimators: 40  Random\_state = 0 |
| **5** | * Null:  Filled with mean/median * Only the top 4 features considered * Outliers treated | Random Forest | 0.844 | N-estimators - 40  Random\_state = 0 |
| **6** | * Null:  Filled with mean/median * Only the top 4 features considered * Outliers treated | Random Forest | 0.8074 | N-estimators - 40  Random state = 42 |
| **7** | * Null:  Filled with mean/median * Only the top 4 features considered * Outliers treated | Random Forest | 0.8000 | N-estimators - 10  Random state = 42 |

**Table 3 – Random Forest Experiment Results**

I attempted to apply the Random Forest model to our dataset, experimenting with various hyperparameter values such as n\_estimators, max\_depth, max\_features, and random\_state. The outcomes of these experiments are presented in the table below. Notably, the highest accuracy score was achieved when n\_estimators was set to 40, and random\_state was set to 0. This accuracy score was the most substantial and comparable to the results obtained using post-pruning techniques, reaching 0.8444.

Additionally, the exploration of various hyperparameter combinations underscored the sensitivity of the Random Forest model to changes in n\_estimators and random\_state. The peak accuracy at n\_estimators = 40 and random\_state = 0 implies that a moderate number of trees, combined with a specific random seed, resulted in the most favorable predictive performance. These insights highlight the intricate interplay between hyperparameter choices and the model's ability to capture underlying patterns in the dataset, emphasizing the importance of thoughtful tuning for robust and accurate predictions.

**(iv) ADABoost:**

ADABoost is one of the decision-tree based model that we use to predict the loan approval decisions, and it only use one learning classifier to train models with reweighted data. The reason we choose the model because it is adaptive to various weak learner that are simple and efficient. Furthermore, ADABoost is a suitable tool to prevent the overfitting problem by combining all the weak learners to create a more robust and generalized model.  Our benchmark for ADAboost, running with dropping the null values, has reached 0.78 for its test accuracy. However, after we switched the null values to mean, median, and mode, the test accuracy are varied from 0.74 - 0.81 and the mean accuracy is 0.79; hence, ADABoost can be a higher accurate model when we apply the data processing techniques.

The hyperparameters we were able to tune when using ADABoost were max\_depth, which controlled the size of the tree for each weak learner in the ensemble, n\_estimators, which defined how many trees to use, and learning\_rate, which affected how much weight each weak learner was given. We found the best results with:

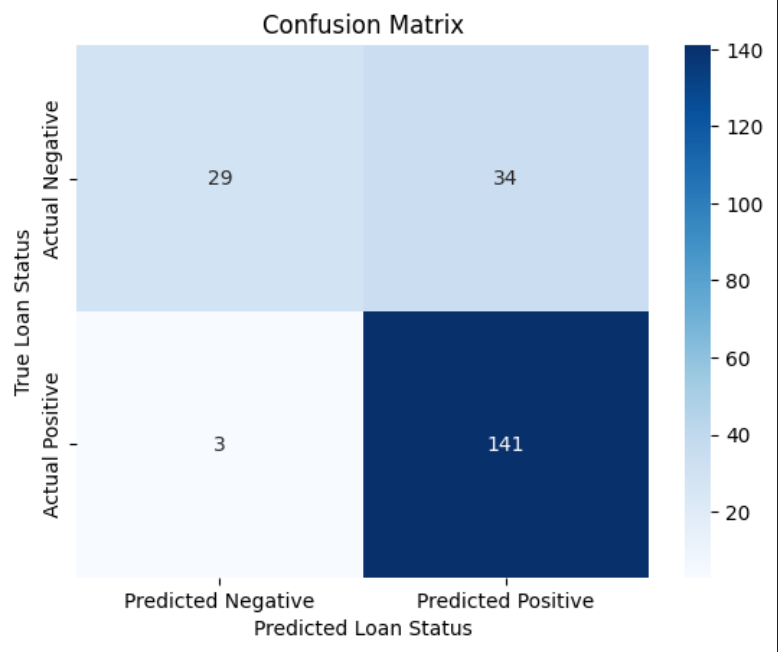
* max\_depth set to 1 - any higher caused the model to perform worse.
* n\_estimators set to between 30 and 150 – using 200 or more worsened the accuracy
* learning\_rate set to 0.05 - making it smaller had no effect, and making it larger reduced the accuracy

The table below shows our results trying various preprocessing methods after finding the hyperparameter tuning that worked best.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Pre-processing** | **Type** | **Train Score** | **Test Score** | **Hyperparameters** |
| 1. | * All null values removed | ADABoost |  | 0.804 | N\_estimators = 100  Learning\_rate = 0.05  Max depth =  1 |
| 2. | * All null values removed * Outliers handled through log transformation + threshold | ADABoost |  | 0.8107 | N\_estimators = 100  Learning\_rate = 0.05  Max depth =  1 |
| 3. | * All null values removed * Binning applied | ADABoost |  | .808 | N\_estimators = 100  Learning\_rate = 0.05  Max depth =  1 |
| 4. | * Nulls replaced with median/mode | ADABoost |  | 0.8095 | N\_estimators = 100  Learning\_rate = 0.05  Max depth =  1 |
| 5. | * Nulls replaced with median/mode * Outliers handled through log transformation + threshold | ADABoost |  | 0.8142 | N\_estimators = 100  Learning\_rate = 0.05  Max depth =  1 |
| 6. | * Nulls replaced with median/mode * Binning applied | ADABoost |  | 0.8096 | N\_estimators = 100  Learning\_rate = 0.05  Max depth =  1 |
| 7. | * Null: Filled with mean/median * Binning applied * Outliers handled through log transformation + threshold | ADABoost |  | 0.8142 | N\_estimators = 100  Learning\_rate = 0.05  Max depth =  1 |
| 8. | * All null values removed * Proportional sampling applied (60/40) | ADABoost |  | 0.768 | N\_estimators = 100  Learning\_rate = 0.05  Max depth =  1 |
| 9. | * All null values removed * Outliers handled through log transformation + threshold * Proportional sampling applied (60/40) | ADABoost |  | 0.74 | N\_estimators = 100  Learning\_rate = 0.05  Max depth =  1 |
| 10. | * Null: Filled with mean/median * Proportional sampling applied (60/40) | ADABoost |  | 0.745 | N\_estimators = 100  Learning\_rate = 0.05  Max depth =  1 |
| 11. | * Null: Filled with mean/median * Proportional sampling applied (60/40) * Outliers handled through log transformation + threshold | ADABoost |  | .760 | N\_estimators = 100  Learning\_rate = 0.05  Max depth =  1 |

**Table 4 – ADABoost Experiment Results**

With these hyperparameters, we got the best results for ADABoost when replacing the nulls with median/mode and handling outliers with the log transformation + threshold, and doing these steps combined with binning had equal performance, resulting in a 10-fold cross validated accuracy of 0.8142.



**Figure 17 - Confusion Matrix for ADABoost under best conditions**

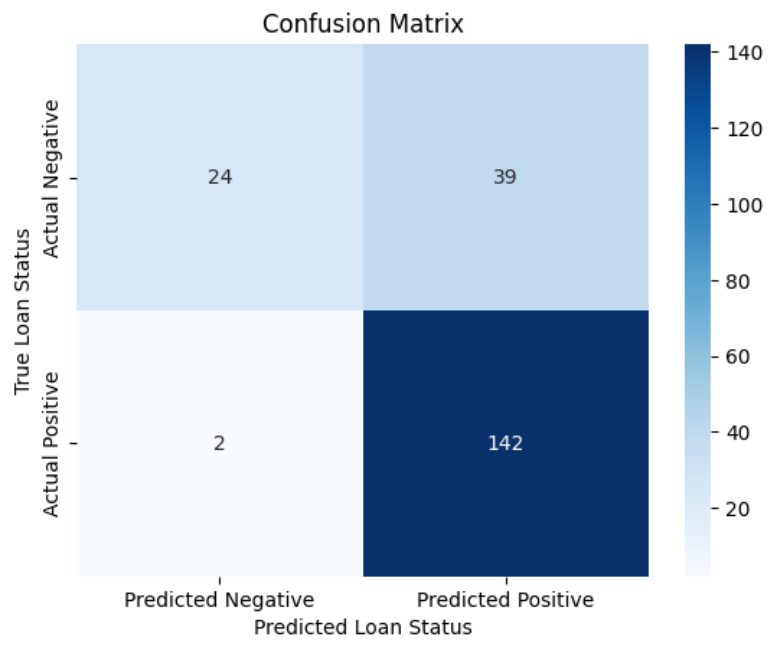
As shown in the above confusion matrix, the model was great at classifying true positive cases as positive but struggled to identify true negative cases. As is, it could be used to filter out applicants who do not qualify for loans, but its predictions when saying an applicant does qualify for a loan is unreliable.

**(v) XGBoost:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Pre-processing** | **Type** | **Train Score** | **Test Score** | **Hyperparameters** |
| 1. | * All null values removed | XGBoost |  | 0.76 | N\_estimators = 100  Learning\_rate = 0.05  Max depth =  2 |
| 2. | * All null values removed * Outliers handled through log transformation + threshold | XGBoost |  | 0.783 | N\_estimators = 100  Learning\_rate = 0.05  Max depth =  2 |
| 3. | * All null values removed * Binning applied | XGBoost |  | .808 | N\_estimators = 100  Learning\_rate = 0.05  Max depth =  2 |
| 4. | * Nulls replaced with median/mode | XGBoost |  | 0.8095 | N\_estimators = 100  Learning\_rate = 0.05  Max depth =  2 |
| 5. | * Nulls replaced with median/mode * Outliers handled through log transformation + threshold | XGBoost |  | 0.8142 | N\_estimators = 100  Learning\_rate = 0.05  Max depth =  2 |
| 6. | * Nulls replaced with median/mode * Binning applied | XGBoost |  | 0.8096 | N\_estimators = 100  Learning\_rate = 0.05  Max depth =  2 |
| 7. | * Null: Filled with mean/median * Binning applied * Outliers handled through log transformation + threshold | XGBoost |  | 0.8142 | N\_estimators = 100  Learning\_rate = 0.05  Max depth =  2 |
| 8. | * All null values removed * Proportional sampling applied (60/40) | XGBoost |  | 0.758 | N\_estimators = 100  Learning\_rate = 0.05  Max depth =  2 |
| 9. | * All null values removed * Outliers handled through log transformation + threshold * Proportional sampling applied (60/40) | XGBoost |  | 0.736 | N\_estimators = 100  Learning\_rate = 0.05  Max depth =  2 |
| 10. | * Null: Filled with mean/median * Proportional sampling applied (60/40) | XGBoost |  | 0.745 | N\_estimators = 100  Learning\_rate = 0.05  Max depth =  2 |
| 11. | * Null: Filled with mean/median * Proportional sampling applied (60/40) * Outliers handled through log transformation + threshold | XGBoost |  | .752 | N\_estimators = 100  Learning\_rate = 0.05  Max depth =  2 |

**Table 5 – XGBoost Experiment Results**

XGBoost has the same parameters as ADABoost, and performed equally well at its peak performance under the same preprocessing conditions and with the same hyperparameter tuning, with the only notable difference being that it performed better with a max depth of 2. a max depth of 1 had reduced accuracy, as did a max depth of 3. Overall, it seemed that as ADABoost and XGBoost are both boosting ensemble models based on decision trees, they perform similarly under similar conditions.



**Figure 18 - Confusion Matrix for XGBoost under best conditions**

As can be seen by the confusion matrix for XGBoost, it performed in a near identical fashion as ADABoost, being good at predicting true positive cases but struggling with true negative cases.

**5. What we learned/conclusions**

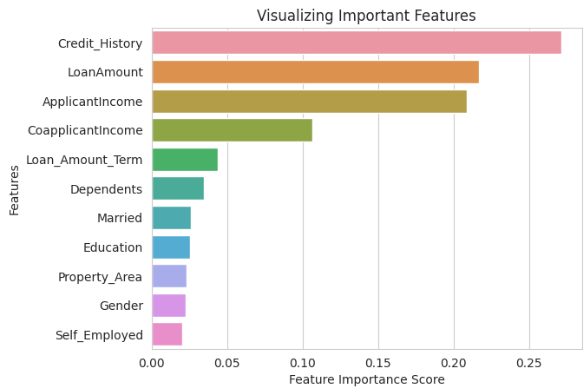
Overall, we have learned the best predictive model for the home loan approval is random forest after pruning, which the accuracy reaches 0.844 mentioned above.

When looking at what we have learned about analysing business problems using machine learning models, we have learned that building good predictive models involve a lot of trial and error. There are many different methods of data preprocessing and many different options for hyperparameter tuning, and while we have an intuition as to what types of preprocessing methods and hyperparameter tuning will produce improved results, sometimes the models behave differently from how we would expect, and as such trying many different methodologies is necessary when building a predictive model

This report focuses on predicting home loan eligibility using data obtained from the Dream Home Financing company website. Our exploration began by assessing data quality, investigating duplicates, unique values, and the distribution of data. We continued with a comprehensive analysis including identifying null values, calculating mean, median, and mode for each column, and conducting univariate and bivariate analyses to understand feature relationships.

Notably, outlier detection became a crucial step in our preprocessing journey, with certain features exhibiting significant deviations. Employing threshold techniques and variable transformations helped manage outliers effectively. Various preprocessing strategies were applied, ranging from removing null values to imputing them with mean and median values. Additionally, we found that it is crucial to perform data preprocessing because of how severely it can raise the accuracy of our predictive models. For example, our Random Forest model accuracy increased 4.4%, 6.4%, and 2.2% when controlling outliers, selecting features, and replacing the null values respectively.

Furthermore, we found through this analysis that when determining whether an applicant is approved for a loan or not, the key features to consider are credit history, loan amount, applicant income, and Co-applicant income as stated in Figure 19. Finally, through analysing the data, we also found out that combining multiple weak learning classifiers in small dataset or those have higher importance on its single feature may not be an ideal predictive model because it may cause a high error on the test data.



**Figure 19. Feature Importance Score**

To predict home loan eligibility, we employed diverse machine learning models, including decision tree, random forest, AdaBoost, XGBoost, and logistic regression. Through 30 experiments involving hyperparameter tuning, we discovered that Random Forest and Decision Tree models outperformed others. The meticulous exploration of hyperparameter combinations yielded optimal settings for our dataset.

In conclusion, our findings underscore the importance of thorough data exploration, effective preprocessing techniques, and careful model selection in predicting home loan eligibility. The successful performance of Random Forest and Decision Tree models, coupled with the identification of best-fit hyperparameters, position our predictive model as a valuable tool for Dream Home Financing in assessing loan eligibility. Using a post-pruned Decision Tree model or a Random Forest model, the loan approval process can likely be automated, as these models had very high accuracy and notably made very type 1 errors. With little to no false positives coming from this model, it can reliably be used, and while some applicants may be turned down even when they should be approved, there will be very few costly mistakes of approving applicants who should not be approved for loans. Automating this process will allow for Dream Housing Finance to free up resources and workforce for other tasks, improving efficiency and in turn leading to increased profit.

References:

Cover Page Images:

N-iX. (n.d.). Artificial Intelligence and machine learning development - N-IX. N. https://www.n-ix.com/machine-learning-ai/

Home loan eligibility and its benefits. idfcfirstbank. (n.d.). https://www.idfcfirstbank.com/finfirst-blogs/home-loan/home-loan-eligibility-and-benefits

Dataset:

Ufffnick. (2019, July 22). Loan prediction dream housing finance. Kaggle. https://www.kaggle.com/code/ufffnick/loan-prediction-dream-housing-finance/input

  (This Kaggle user obtained the Dataset from Analytics Vidhya’s 2016 ML contest):

Loan prediction. Analytics Vidhya. (n.d.). https://datahack.analyticsvidhya.com/contest/practice-problem-loan-prediction-iii/