International Institute of
Information Technology Bangalore

AIM 511 - Machine Learning

Project Report

Lend or Lose

Authors: Shashank
Devarmani
IMT2022107
Pradyun Devarakonda
IMT2022525
Shanmukh Praneeth
IMT2022542

GitHub Repository

The complete set of codes, notebooks, and related resources for this project can be found at the following GitHub repository: `https://github.com/standing-on-giants/Lend-or-lose-Loan-Default-prediction-project`

Problem Statement

Financial loan services are leveraged by companies across many industries, from big banks to financial institutions to government loans. One of the primary objectives of companies with financial loan services is to decrease payment defaults and ensure that individuals are paying back their loans as expected. To do this efficiently and systematically, many companies employ machine learning to predict which individuals are at the highest risk of defaulting on their loans, so that proper interventions can be effectively deployed to the right audience.This dataset has been taken from Coursera's Loan Default Prediction Challenge.

Data Preprocessing

- **Null Values:**

Dealing with null values

```
#Check %tage of null values
train_df.isnull().sum()
```

```
Age               0
Income            0
LoanAmount        0
CreditScore       0
MonthsEmployed    0
NumCreditLines    0
InterestRate      0
LoanTerm          0
DTIRatio          0
Education         0
EmploymentType    0
MaritalStatus     0
HasMortgage       0
HasDependents     0
LoanPurpose       0
HasCoSigner       0
Default           0
dtype: int64
```

There were no null values present in the training dataset so there is no need to remove them.

- **Outliers:**

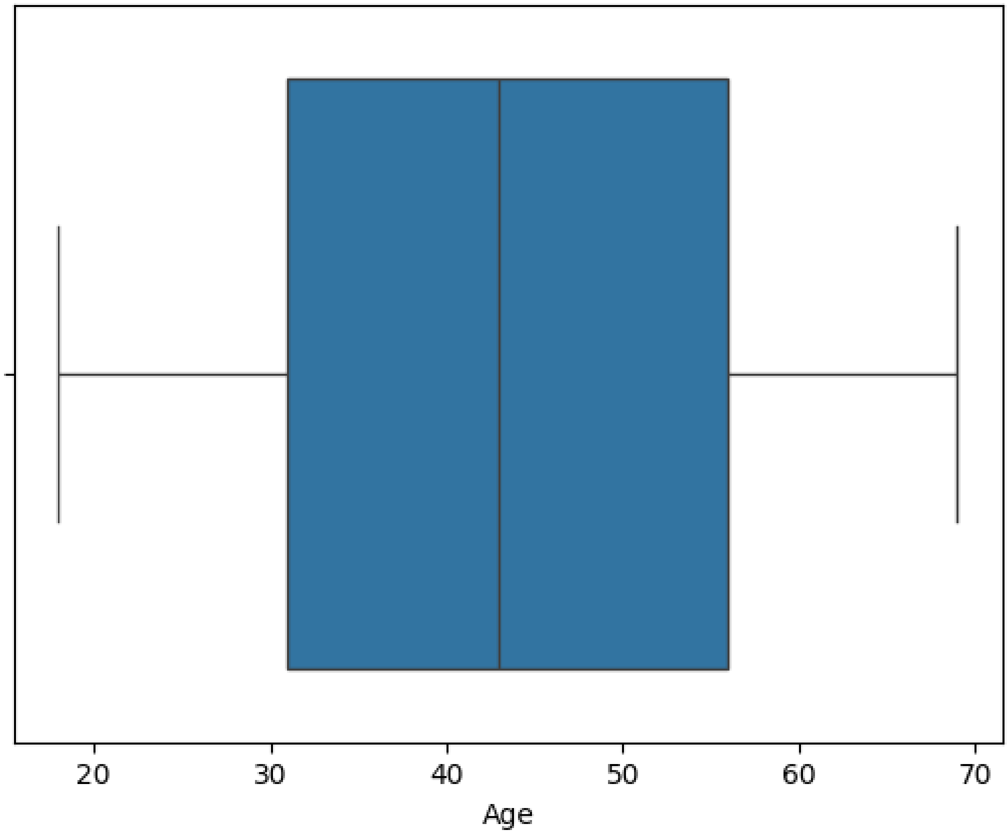


Figure 1: Age: Boxplot

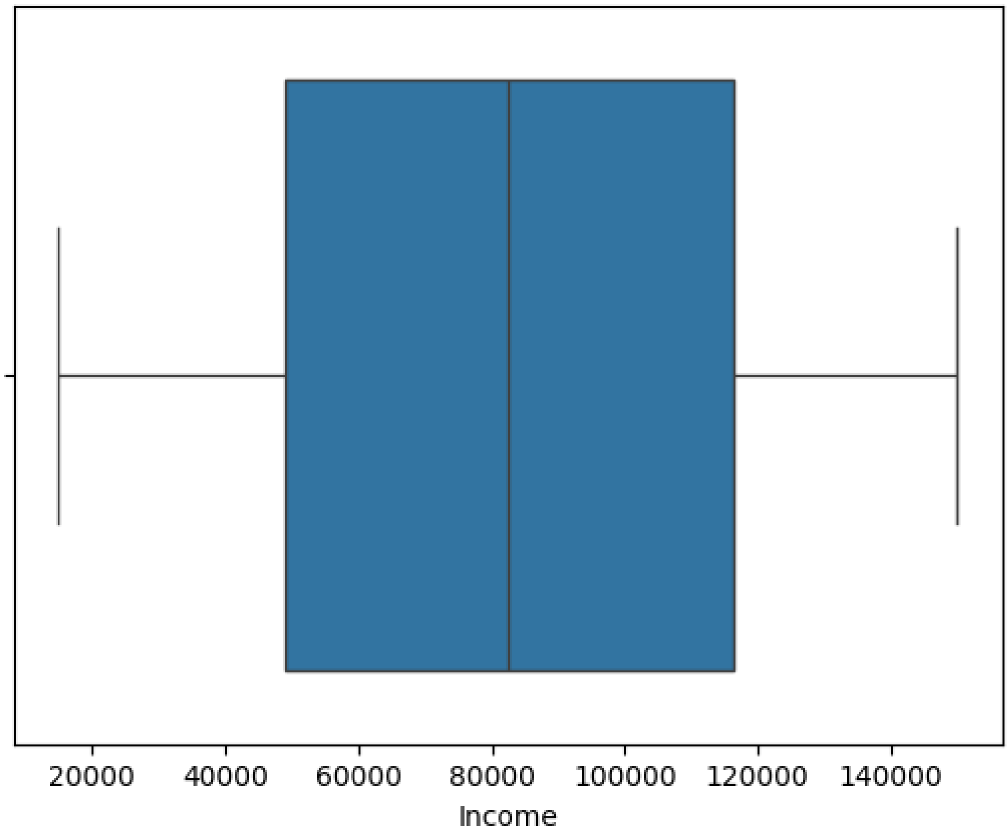


Figure 2: Income: Boxplot

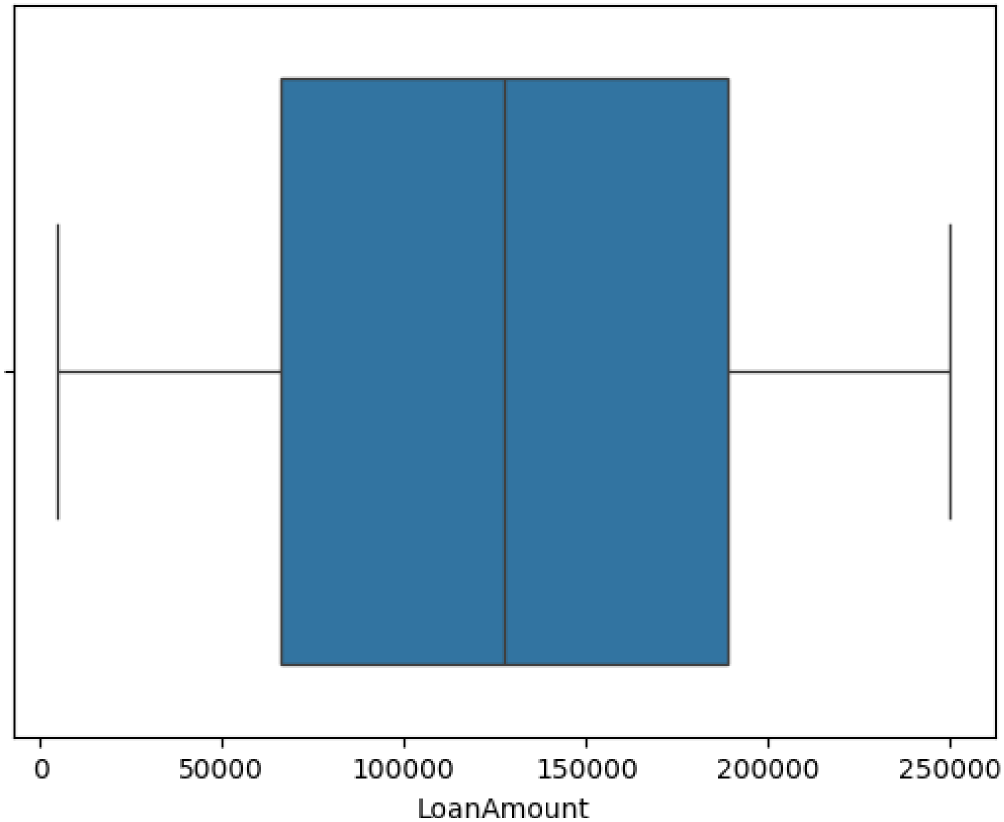


Figure 3: LoanAmount: Boxplot

There were no outliers to remove which is evident from the box plots above. Also, decision trees are robust to outliers.

- **One-Hot Encoder:**

Categorical to numerical Encoding

```python
# First, identify categorical and numerical columns
from sklearn.preprocessing import OneHotEncoder

categorical_columns_train = train_df.select_dtypes(include=['object']).columns
numerical_columns_train = train_df.select_dtypes(include=['int64', 'float64']).columns

categorical_columns_test = test_df.select_dtypes(include=['object']).columns
numerical_columns_test = test_df.select_dtypes(include=['int64', 'float64']).columns

print("Categorical columns:", categorical_columns_train.tolist())
print("Numerical columns:", numerical_columns_train.tolist())

# Initialize the encoder
encoder = OneHotEncoder(sparse_output=False)
encoder1 = OneHotEncoder(sparse_output=False)

# Fit and transform only the categorical columns
categorical_encoded_train = encoder.fit_transform(train_df[categorical_columns_train])
categorical_encoded_test = encoder1.fit_transform(test_df[categorical_columns_train])

# Create DataFrame with encoded categorical variables
encoded_categorical_traindf = pd.DataFrame(
    categorical_encoded_train,
    columns=encoder.get_feature_names_out(categorical_columns_train)
)
encoded_categorical_testdf = pd.DataFrame(
    categorical_encoded_test,
    columns=encoder1.get_feature_names_out(categorical_columns_test)
)

# Combine with numerical columns
encoded_train_df = pd.concat([
    encoded_categorical_traindf,
    train_df[numerical_columns_train].reset_index(drop=True)
], axis=1)


encoded_test_df = pd.concat([
    encoded_categorical_testdf,
    test_df[numerical_columns_test].reset_index(drop=True)
], axis=1)

encoded_test_df
```

[87]

```
Categorical columns: ['Education', 'EmploymentType', 'MaritalStatus', 'HasMortgage', 'HasDependents', 'LoanPurpose', 'HasCoSigner']
Numerical columns: ['Age', 'Income', 'LoanAmount', 'CreditScore', 'MonthsEmployed', 'NumCreditLines', 'InterestRate', 'LoanTerm', 'DTIRatio', 'Default']
```

We used one-hot encoding to transform categorical variables into binary format. Unlike label encoding, which assigns numerical values to categories, one-hot encoding prevents the model from interpreting categories like "Single," "Married," and "Divorced" as having an ordinal relationship. This approach ensures the model treats each category as distinct and unrelated, which is crucial for algorithms sensitive to numeric relationships.

EDA (Exploratory Data Analysis)

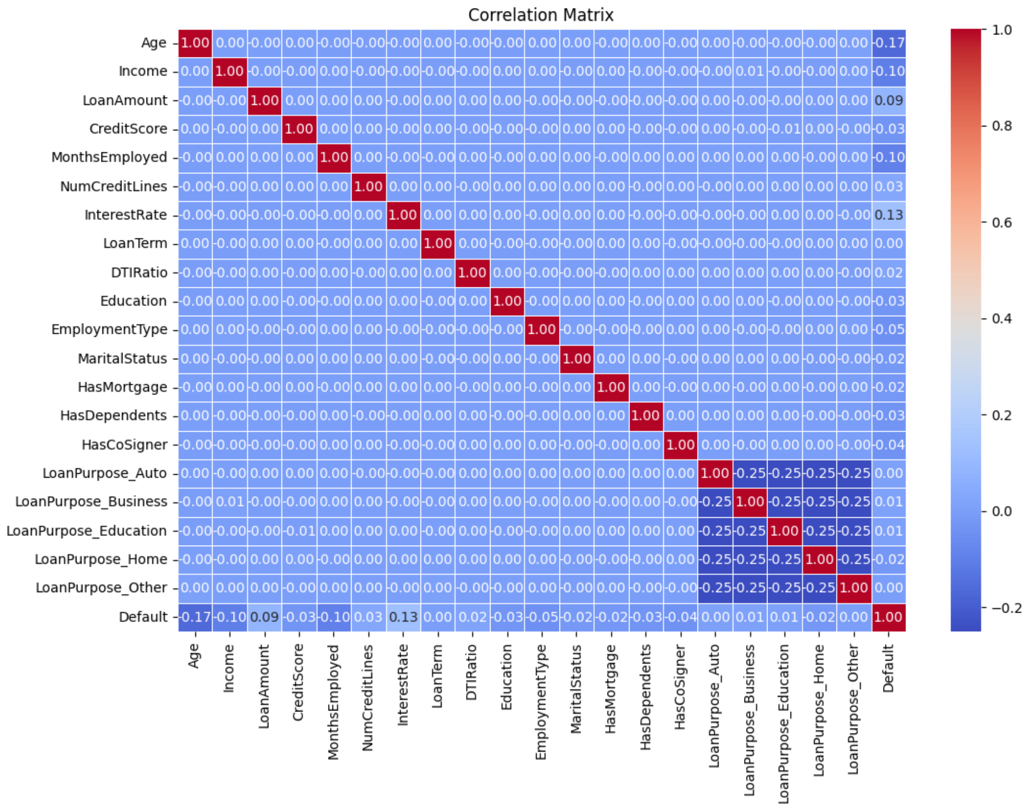


Figure 4: Correlation Matrix

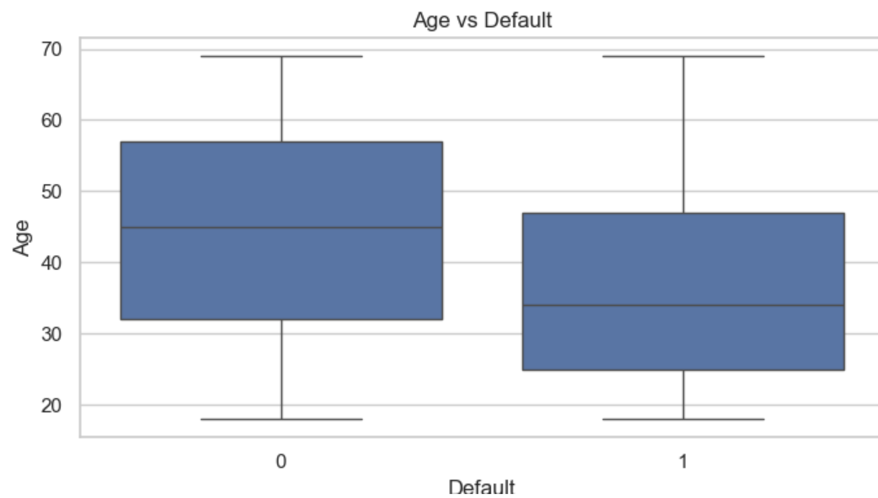


Figure 5: Age vs Default

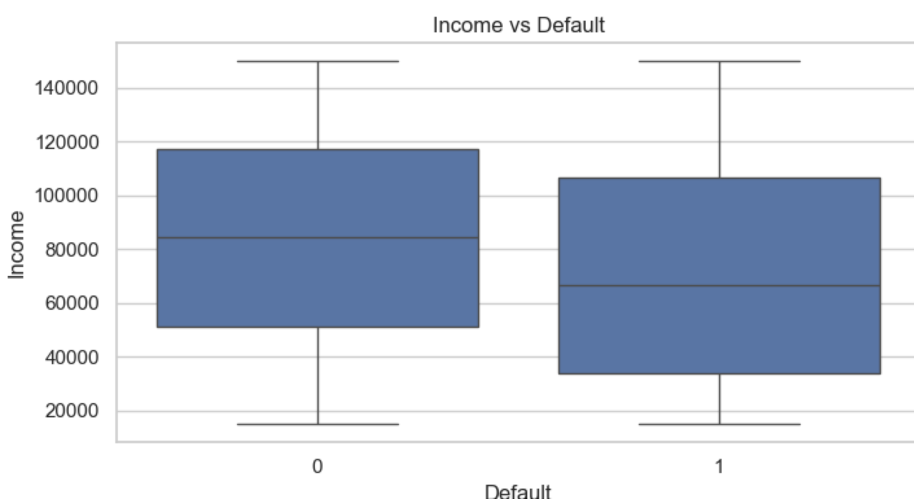


Figure 6: Income vs Default

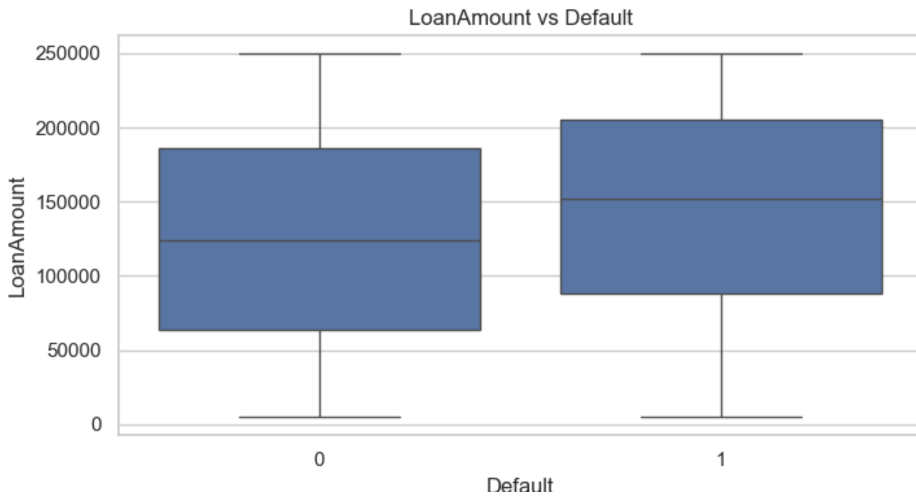


Figure 7: LoanAmount vs Default

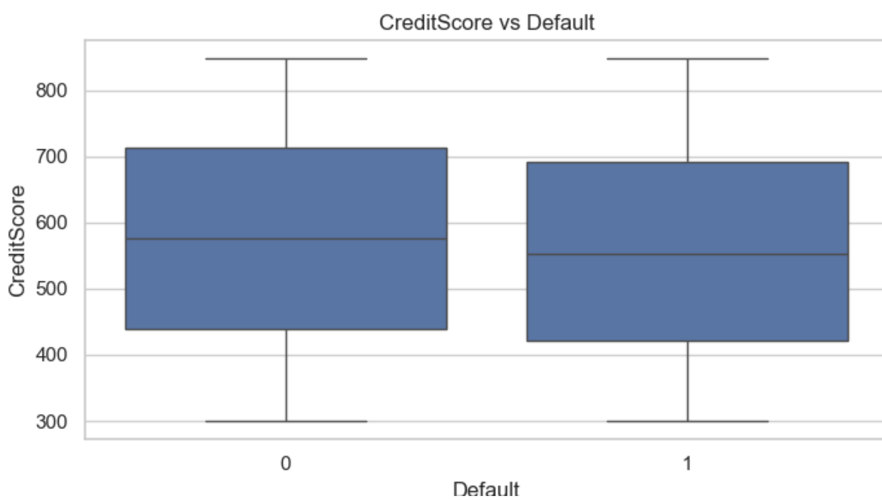


Figure 8: CreditScore vs Default

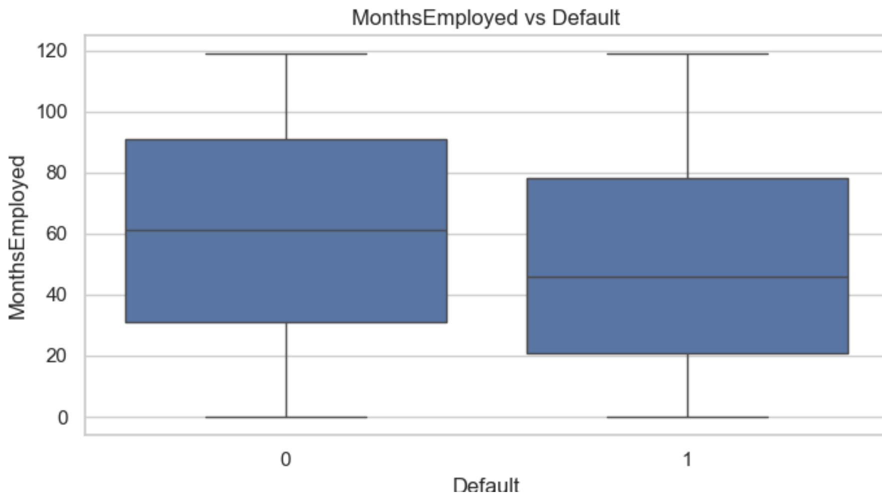


Figure 9: MonthsEmployed vs Default

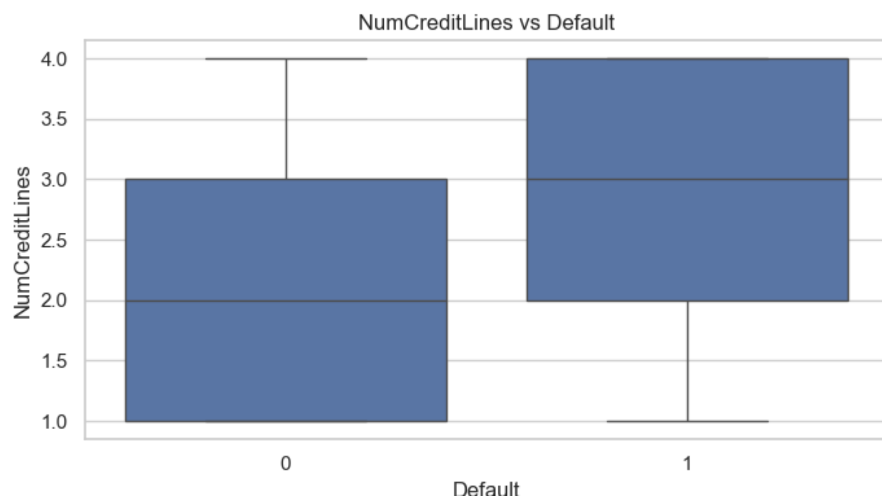


Figure 10: NumCreditLines vs Default

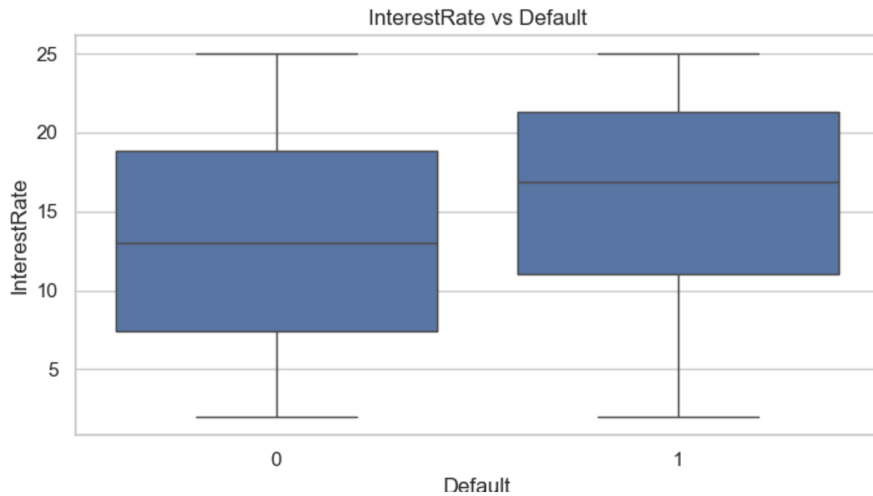


Figure 11: InterestRate vs Default

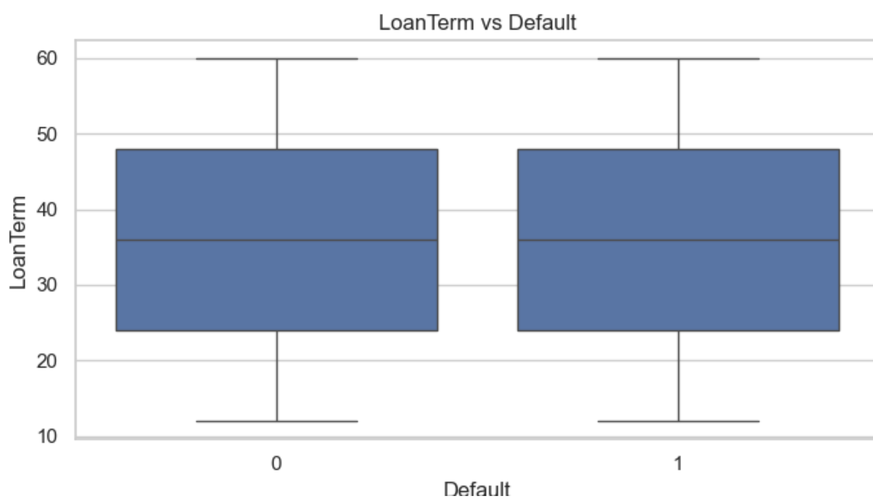


Figure 12: LoanTerm vs Default

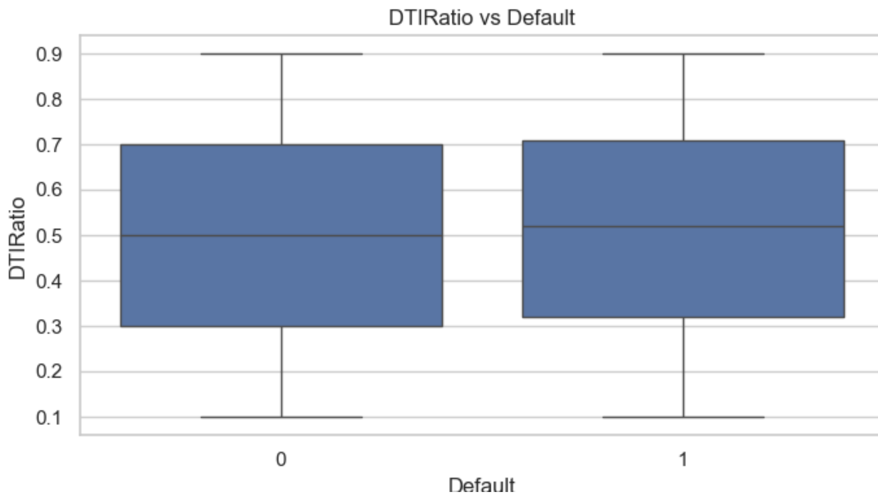


Figure 13: DTIRatio vs Default

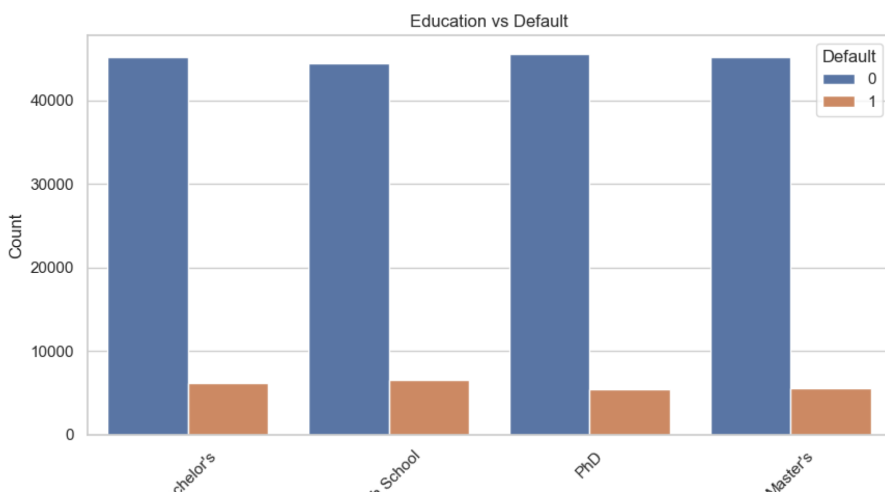


Figure 14: Education vs Default

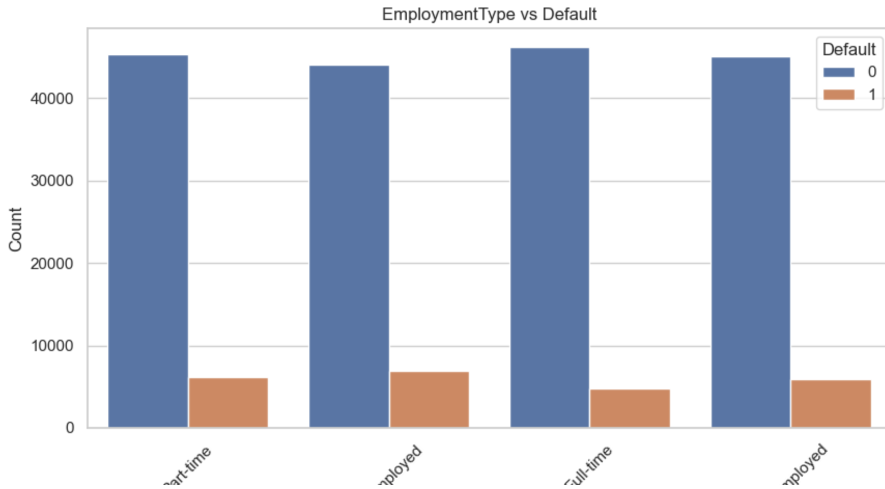


Figure 15: EmploymentType vs Default

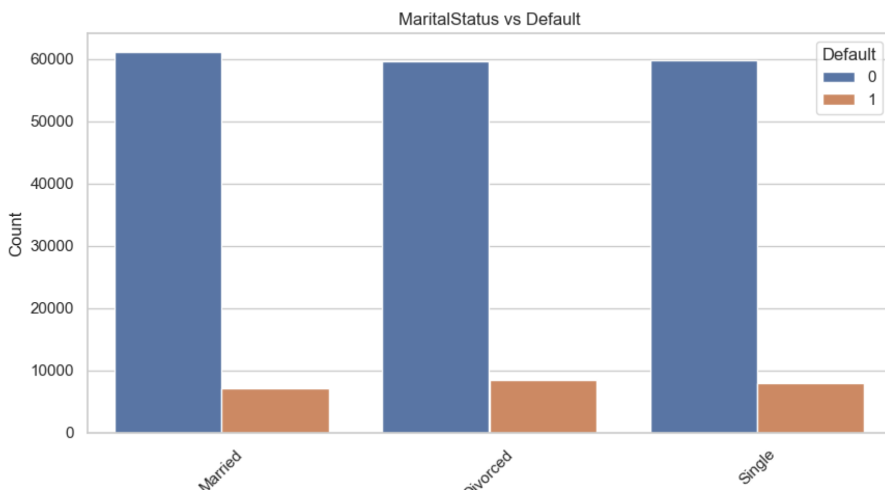


Figure 16: MaritalStatus vs Default

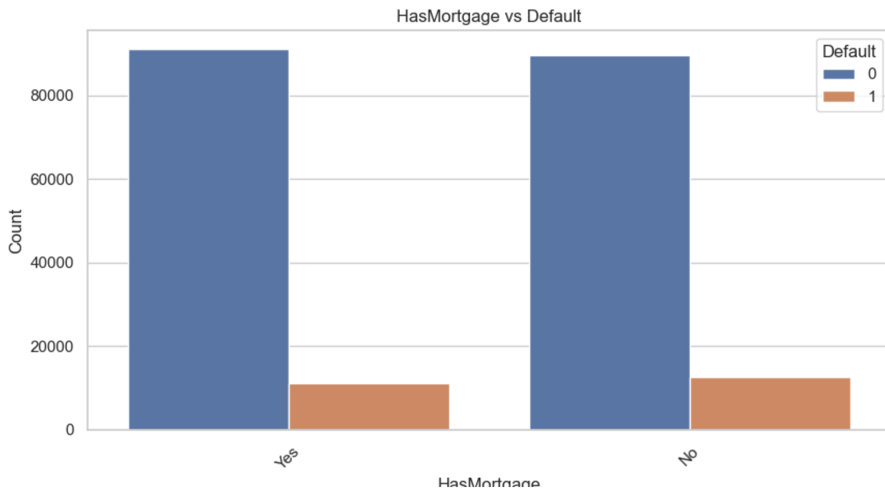


Figure 17: HasMortgage vs Default

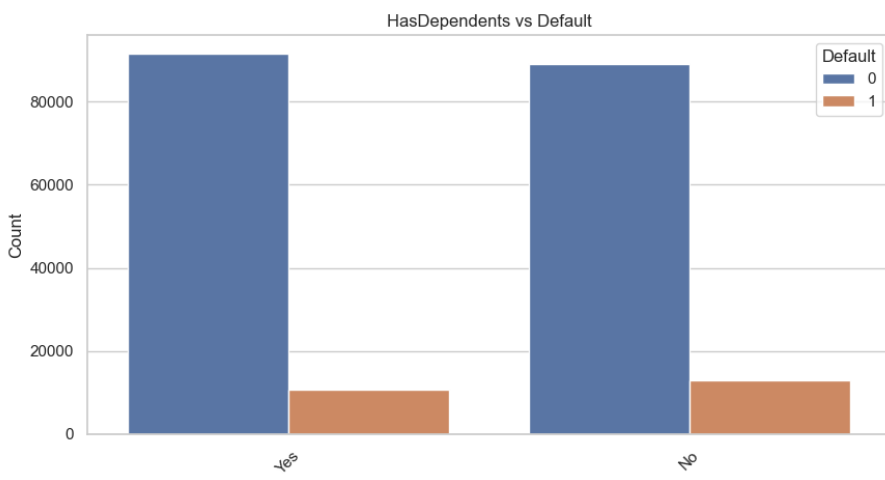


Figure 18: HasDependents vs Default

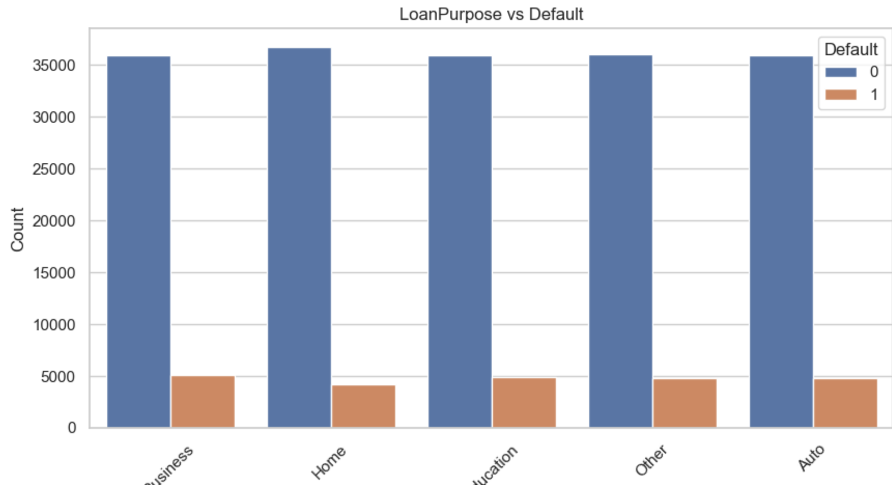


Figure 19: LoanPurpose vs Default

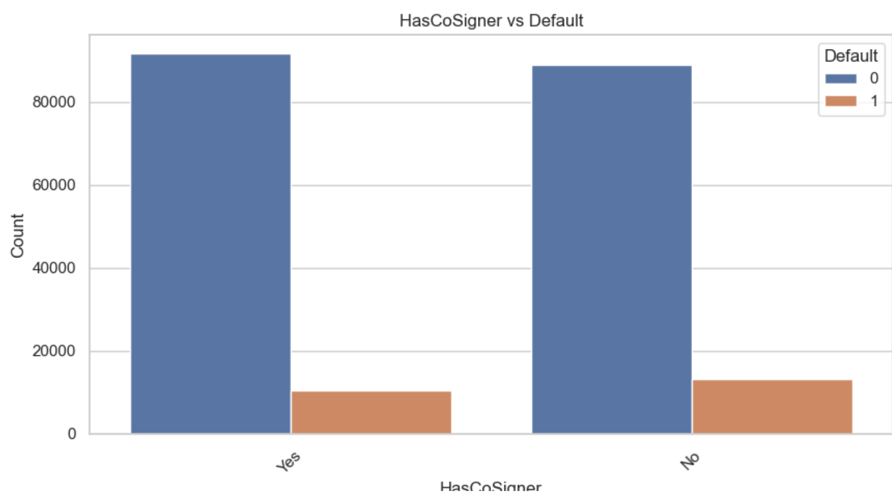


Figure 20: HasCoSigner vs Default

We dropped the LoanID column since that column is just for indexing and shouldn't be used for predicting our target variable. We used all the columns since they were important and had some correlation with our target variable 'Default'.

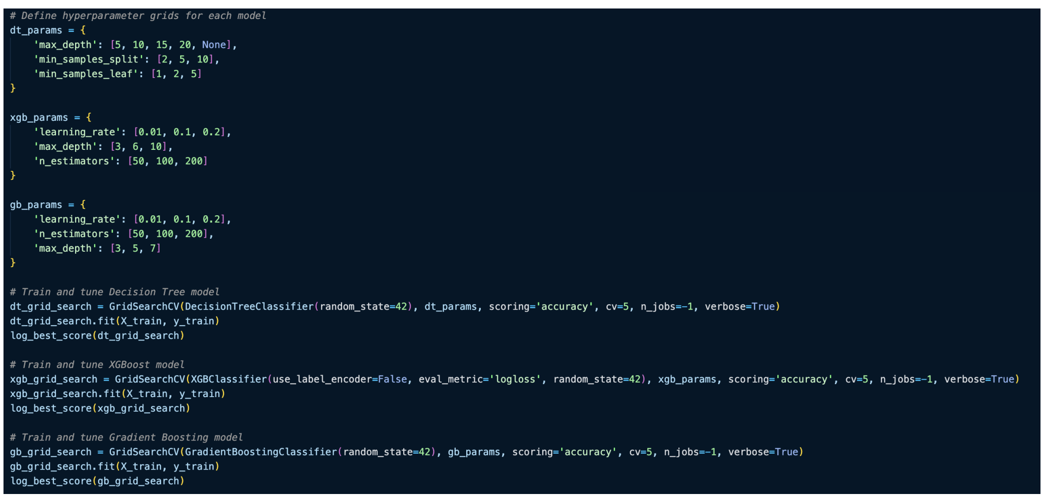
Models Used

We have utilized the following models in our analysis:

- **Decision Trees**: A tree-based model that splits the data recursively to predict the target variable based on feature thresholds. This model is great
- **Gradient Boosting**: An ensemble method that builds sequential decision trees, where each tree corrects the errors of its predecessor, aiming to minimize the loss function.
- **Gaussian Naive Bayes**: A probabilistic model based on Bayes' theorem, assuming normal distribution for continuous features.
- **Bernoulli Naive Bayes**: A variant of Naive Bayes suited for binary/boolean features.
- **Multinomial Naive Bayes**: Best suited for multinomially distributed data, often used in text classification.
- **K-Nearest Neighbors (KNN)**: A distance-based algorithm that predicts the target class based on the majority class among the nearest neighbors.
- **XGBoost - Best Model**: An optimized gradient boosting framework that enhances prediction performance through efficient parallelization.

Hyperparameter Tuning

To optimize hyperparameters for some of the above models, we employed GridSearchCV cross-validation methods. While the returned hyperparameter values marginally improved the validation score, we ultimately chose to use only select hyperparameter values after carefully weighing the trade-off with training time.



```python
# Define hyperparameter grids for each model
dt_params = {
    'max_depth': [5, 10, 15, 20, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5]
}

xgb_params = {
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 6, 10],
    'n_estimators': [50, 100, 200]
}

gb_params = {
    'learning_rate': [0.01, 0.1, 0.2],
    'n_estimators': [50, 100, 200],
    'max_depth': [3, 5, 7]
}

# Train and tune Decision Tree model
dt_grid_search = GridSearchCV(DecisionTreeClassifier(random_state=42), dt_params, scoring='accuracy', cv=5, n_jobs=-1, verbose=True)
dt_grid_search.fit(X_train, y_train)
log_best_score(dt_grid_search)

# Train and tune XGBoost model
xgb_grid_search = GridSearchCV(XGBClassifier(use_label_encoder=False, eval_metric='logloss', random_state=42), xgb_params, scoring='accuracy', cv=5, n_jobs=-1, verbose=True)
xgb_grid_search.fit(X_train, y_train)
log_best_score(xgb_grid_search)

# Train and tune Gradient Boosting model
gb_grid_search = GridSearchCV(GradientBoostingClassifier(random_state=42), gb_params, scoring='accuracy', cv=5, n_jobs=-1, verbose=True)
gb_grid_search.fit(X_train, y_train)
log_best_score(gb_grid_search)
```

Figure 21: Hyperparameter tuning

Validation and Results

Model	Result on Kaggle	Approach
Multinomial Naive Bayes Classifier	0.57918	One Hot Encoding
Decision Trees	0.80072	One Hot Encoding and Hyperparameter Tuning
Bernoulli Naive Bayes Classifier	0.88447	One Hot Encoding
KNN Classifier	0.88447	Normalization and Hyperparameter Tuning
Gaussian Naive Bayes Classifier	0.88488	One Hot Encoding
Gradient Boosting Classifier	0.88707	One-Hot Encoding and Hyperparameter Tuning
XG Boost	0.88752	One-Hot Encoding and Hyperparameter Tuning

Table 1: Model Results and Approaches

Model Performance Analysis

- **Multinomial Naive Bayes:**
  - Achieved moderate performance due to its assumption of categorical data distributions, which may not fully align with the dataset.
- **Decision Trees:**
  - Benefited from hyperparameter tuning and effectively captured non-linear relationships, leading to better results.
- **Bernoulli Naive Bayes:**
  - Performed well by efficiently handling binary features, leveraging its simplicity for high accuracy.
- **Gaussian Naive Bayes:**
  - Achieved high scores due to its ability to handle continuous features effectively and model their distributions.
- **KNN Classifier:**
  - Showed significant improvement after normalization and hyperparameter tuning, emphasizing the importance of scaling for distance-based models.
- **Gradient Boosting and XGBoost:**
  - Outperformed other models by combining multiple weak learners, optimizing complex patterns in the dataset, and leveraging robust hyperparameter tuning.