Final Project

Michael Ritacco

2024-04-14

Table of Contents

# Import Data and Required Libraries

# Import Dependencies  
suppressPackageStartupMessages({  
 library(ggplot2)  
 library(dplyr)  
 library(caret)  
 library(randomForest)  
 library(MLmetrics)  
 library(PRROC)  
 library(xgboost)  
 library(reshape2)  
 library(tidyr)  
 library(parallel)  
 library(doParallel)  
})

## Warning: package 'ggplot2' was built under R version 4.3.1

## Warning: package 'dplyr' was built under R version 4.3.1

## Warning: package 'MLmetrics' was built under R version 4.3.1

## Warning: package 'xgboost' was built under R version 4.3.1

# Get the number of cores  
no\_cores = detectCores() - 1   
# Register the number of cores to parallelize the model fitting  
registerDoParallel(cores = no\_cores)

# Read in the data  
df = read.csv('loan\_default.csv')  
  
# Output the first 5 rows of the dataset  
head(df)

## LoanID Age Income LoanAmount CreditScore MonthsEmployed NumCreditLines  
## 1 I38PQUQS96 56 85994 50587 520 80 4  
## 2 HPSK72WA7R 69 50432 124440 458 15 1  
## 3 C1OZ6DPJ8Y 46 84208 129188 451 26 3  
## 4 V2KKSFM3UN 32 31713 44799 743 0 3  
## 5 EY08JDHTZP 60 20437 9139 633 8 4  
## 6 A9S62RQ7US 25 90298 90448 720 18 2  
## InterestRate LoanTerm DTIRatio Education EmploymentType MaritalStatus  
## 1 15.23 36 0.44 Bachelor's Full-time Divorced  
## 2 4.81 60 0.68 Master's Full-time Married  
## 3 21.17 24 0.31 Master's Unemployed Divorced  
## 4 7.07 24 0.23 High School Full-time Married  
## 5 6.51 48 0.73 Bachelor's Unemployed Divorced  
## 6 22.72 24 0.10 High School Unemployed Single  
## HasMortgage HasDependents LoanPurpose HasCoSigner Default  
## 1 Yes Yes Other Yes 0  
## 2 No No Other Yes 0  
## 3 Yes Yes Auto No 1  
## 4 No No Business No 0  
## 5 No Yes Auto No 0  
## 6 Yes No Business Yes 1

# Data Preprocessing

## Data Cleaning

# Get the column names  
colnames(df)

## [1] "LoanID" "Age" "Income" "LoanAmount"   
## [5] "CreditScore" "MonthsEmployed" "NumCreditLines" "InterestRate"   
## [9] "LoanTerm" "DTIRatio" "Education" "EmploymentType"  
## [13] "MaritalStatus" "HasMortgage" "HasDependents" "LoanPurpose"   
## [17] "HasCoSigner" "Default"

# Removing the index column  
df = df[-1]

## Examining Dataset Structure

# Examine the structure of the dataset  
str(df)

## 'data.frame': 255347 obs. of 17 variables:  
## $ Age : int 56 69 46 32 60 25 38 56 36 40 ...  
## $ Income : int 85994 50432 84208 31713 20437 90298 111188 126802 42053 132784 ...  
## $ LoanAmount : int 50587 124440 129188 44799 9139 90448 177025 155511 92357 228510 ...  
## $ CreditScore : int 520 458 451 743 633 720 429 531 827 480 ...  
## $ MonthsEmployed: int 80 15 26 0 8 18 80 67 83 114 ...  
## $ NumCreditLines: int 4 1 3 3 4 2 1 4 1 4 ...  
## $ InterestRate : num 15.23 4.81 21.17 7.07 6.51 ...  
## $ LoanTerm : int 36 60 24 24 48 24 12 60 48 48 ...  
## $ DTIRatio : num 0.44 0.68 0.31 0.23 0.73 0.1 0.16 0.43 0.2 0.33 ...  
## $ Education : chr "Bachelor's" "Master's" "Master's" "High School" ...  
## $ EmploymentType: chr "Full-time" "Full-time" "Unemployed" "Full-time" ...  
## $ MaritalStatus : chr "Divorced" "Married" "Divorced" "Married" ...  
## $ HasMortgage : chr "Yes" "No" "Yes" "No" ...  
## $ HasDependents : chr "Yes" "No" "Yes" "No" ...  
## $ LoanPurpose : chr "Other" "Other" "Auto" "Business" ...  
## $ HasCoSigner : chr "Yes" "Yes" "No" "No" ...  
## $ Default : int 0 0 1 0 0 1 0 0 1 0 ...

# Get the total number of observations  
n = nrow(df)  
  
# Get the number of columns  
p = ncol(df)  
  
# Output the number of observations and columns   
cat("There are", n, 'observations and ', p, 'columns in our dataset.')

## There are 255347 observations and 17 columns in our dataset.

## Check for Missing Values

# Check for missing values across columns  
colSums(is.na(df))

## Age Income LoanAmount CreditScore MonthsEmployed   
## 0 0 0 0 0   
## NumCreditLines InterestRate LoanTerm DTIRatio Education   
## 0 0 0 0 0   
## EmploymentType MaritalStatus HasMortgage HasDependents LoanPurpose   
## 0 0 0 0 0   
## HasCoSigner Default   
## 0 0

## Encode Categorical Variables

# Define vector of continuous variables  
continuous\_vars = c(  
 'Age', 'Income', 'LoanAmount', 'CreditScore',  
 'MonthsEmployed', 'InterestRate', 'DTIRatio'  
)  
  
# Define vector of categorical variables  
categorical\_vars = c(  
 'Education', 'EmploymentType', 'MaritalStatus',  
 'HasMortgage', 'HasDependents', 'LoanPurpose', 'HasCoSigner'  
)  
  
# Define vector of ordinal variables  
ordinal\_vars = c(  
 'NumCreditLines', 'LoanTerm'  
)  
  
# Define target variable  
target = 'Default'  
  
# Encode categorical variable as a factor  
for (var in c(categorical\_vars, ordinal\_vars)) {  
 df[[var]] = as.factor(df[[var]])  
}  
  
# Encode target variable as a factor  
df$Default = factor(df$Default, levels = c('1', '0'), labels = c('Default', 'NonDefault'))

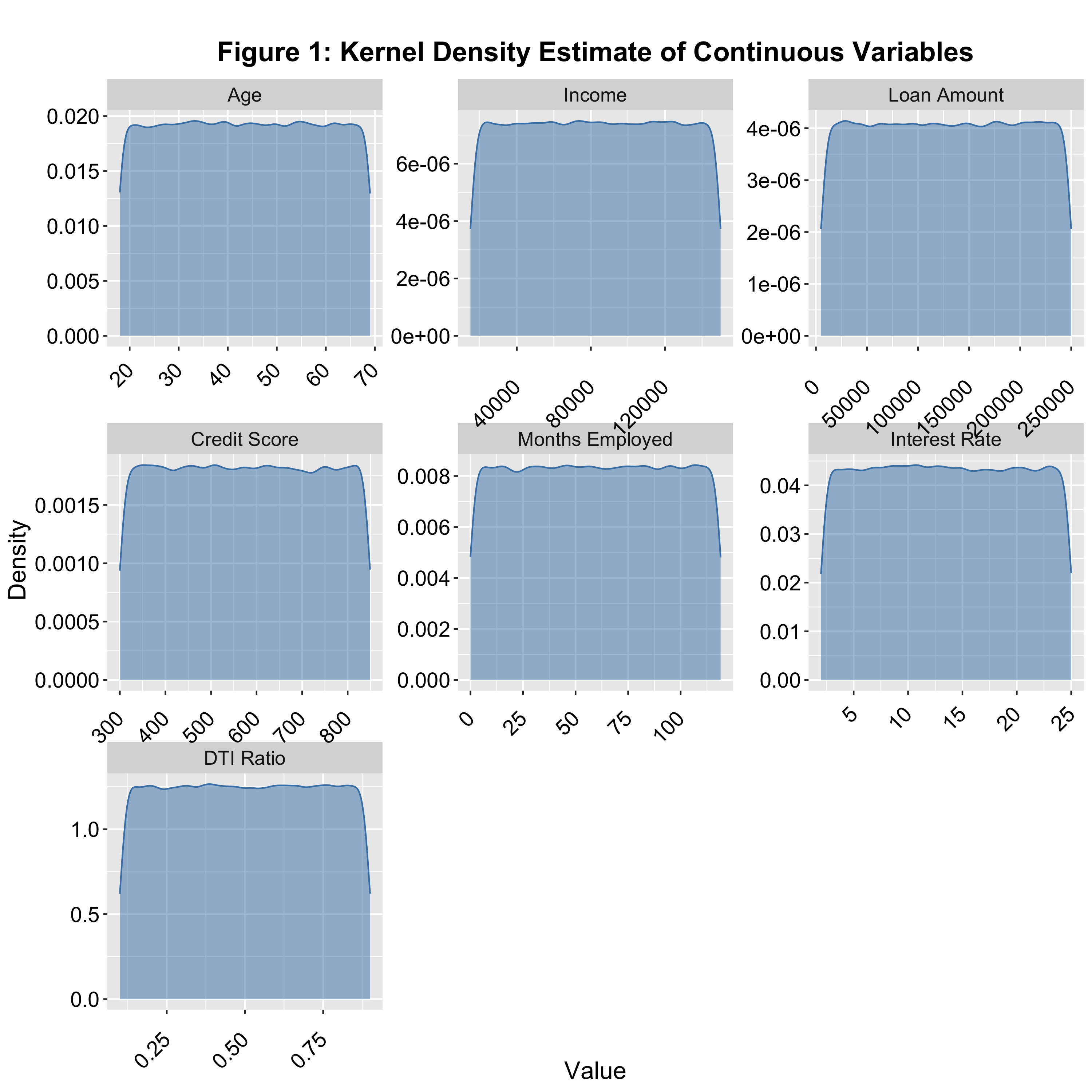
# Exploratory Data Analysis

## Distribution of Continuous Variables

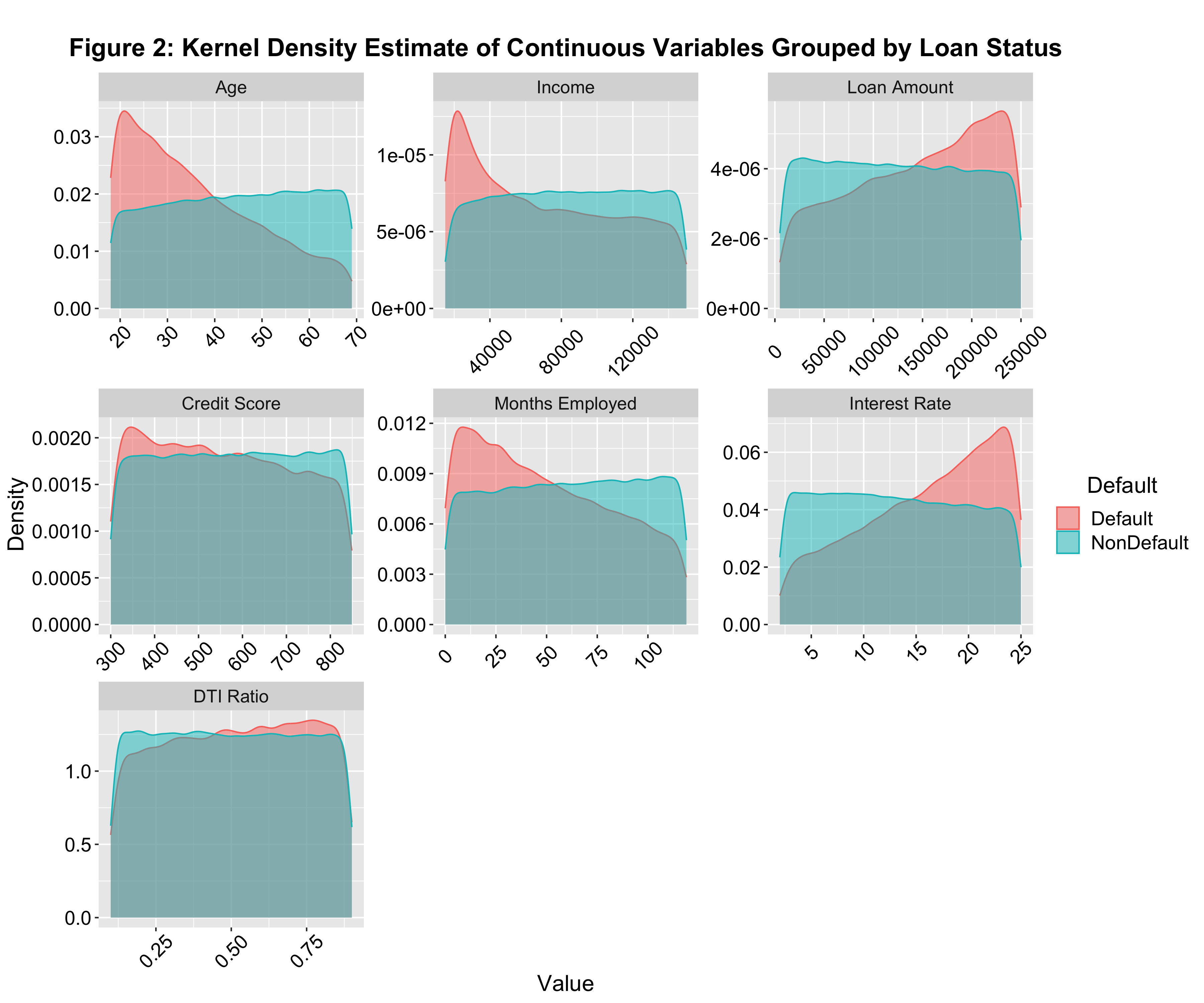
# Calculate the summary statistics  
summary(df[continuous\_vars])

## Age Income LoanAmount CreditScore   
## Min. :18.0 Min. : 15000 Min. : 5000 Min. :300.0   
## 1st Qu.:31.0 1st Qu.: 48826 1st Qu.: 66156 1st Qu.:437.0   
## Median :43.0 Median : 82466 Median :127556 Median :574.0   
## Mean :43.5 Mean : 82499 Mean :127579 Mean :574.3   
## 3rd Qu.:56.0 3rd Qu.:116219 3rd Qu.:188985 3rd Qu.:712.0   
## Max. :69.0 Max. :149999 Max. :249999 Max. :849.0   
## MonthsEmployed InterestRate DTIRatio   
## Min. : 0.00 Min. : 2.00 Min. :0.1000   
## 1st Qu.: 30.00 1st Qu.: 7.77 1st Qu.:0.3000   
## Median : 60.00 Median :13.46 Median :0.5000   
## Mean : 59.54 Mean :13.49 Mean :0.5002   
## 3rd Qu.: 90.00 3rd Qu.:19.25 3rd Qu.:0.7000   
## Max. :119.00 Max. :25.00 Max. :0.9000

# Define custom theme for plotting  
my\_custom\_theme = theme(  
 text = element\_text(size = 16), # Sets global text size for all text elements  
 plot.title = element\_text(hjust = 0.5, size = 18, face = "bold", margin = ggplot2::margin(t = 20, b = 10)), # Specific title adjustments  
 axis.title = element\_text(size = 16), # Axis titles  
 axis.text = element\_text(size = 14), # Axis text  
 legend.title = element\_text(size = 16), # Legend title  
 legend.text = element\_text(size = 14) # Legend text  
 )  
  
# Set custom theme as the default for all subsequent plots  
theme\_set(my\_custom\_theme)  
  
# Reshape the dataframe to long format for plotting  
df\_long = melt(df, measure.vars = continuous\_vars)  
  
# Create a named vector to map variable names to proper labels when plotting  
var\_labels = c(  
 Age = 'Age', Income = 'Income', LoanAmount = 'Loan Amount',   
 CreditScore = 'Credit Score', MonthsEmployed = 'Months Employed',  
 NumCreditLines = 'Number of Credit Lines', InterestRate = 'Interest Rate',  
 LoanTerm = 'Loan Term', DTIRatio = 'DTI Ratio', Education = 'Education',   
 EmploymentType = 'Employment Type', MaritalStatus = 'Marital Status',   
 HasMortgage = 'Has Mortgage', HasDependents = 'Has Dependents',  
 LoanPurpose = 'Loan Purpose', HasCoSigner = 'Has Co Signer',   
 Default = 'Loan Status'  
)  
  
# Generate KDE plots for each continuous variable  
ggplot(df\_long, aes(x = value)) +  
 geom\_density(color = 'steelblue', fill = 'steelblue', alpha = 0.5) +   
 facet\_wrap(~ variable, scales = 'free', ncol = 3,  
 labeller = labeller(variable = var\_labels)) +   
 labs(title = 'Figure 1: Kernel Density Estimate of Continuous Variables',  
 x = 'Value', y = 'Density') +  
 theme(axis.text.x = element\_text(angle = 45, hjust = 1),   
 plot.title = element\_text(hjust = 0.5))



# Generate KDE plots for each continuous variable grouped by loan status  
ggplot(df\_long, aes(x = value, color = Default, fill = Default)) +  
 geom\_density(alpha = 0.5) +   
 facet\_wrap(~ variable, scales = 'free',   
 labeller = labeller(variable = var\_labels), ncol = 3) +   
 labs(title = 'Figure 2: Kernel Density Estimate of Continuous Variables Grouped by Loan Status',  
 x = 'Value',  
 y = 'Density') +  
 theme(axis.text.x = element\_text(angle = 45, hjust = 0.5))

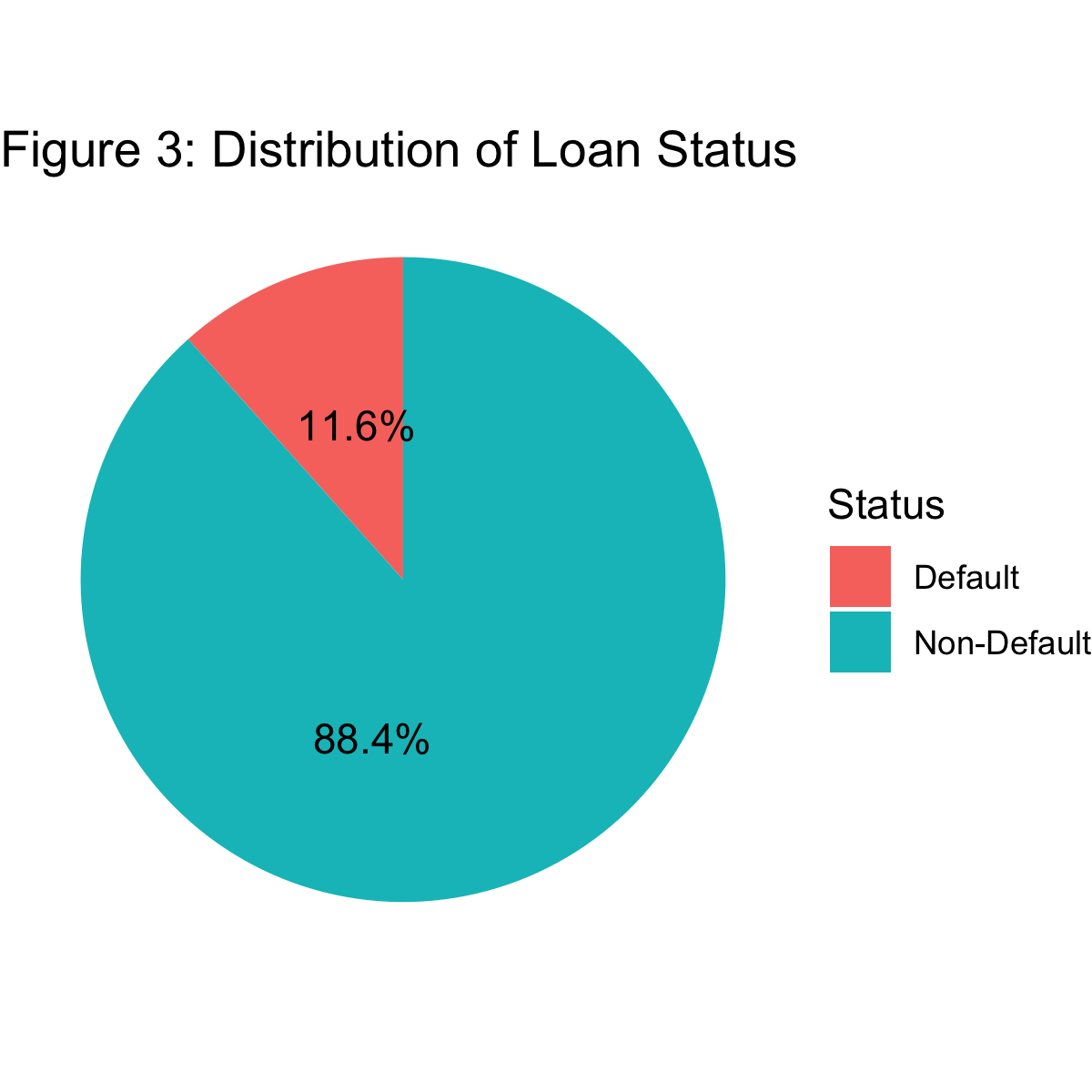


## Distribution of Target Variable

# Calculate counts of the loan default status in the dataframe  
default\_counts = table(df$Default)   
# Calculate proportions of each loan Default status relative to the total  
default\_proportions = (default\_counts / n) \* 100  
  
# Create and output a data frame for plotting, labeling each category and converting counts and proportions to vectors  
default\_df = data.frame(  
 Status = c('Default', 'Non-Default'), # Define categories of loan status  
 Count = as.vector(default\_counts), # Include counts of each category  
 Percent = as.vector(default\_proportions) # Include percentage of each category  
 )  
default\_df

## Status Count Percent  
## 1 Default 29653 11.61282  
## 2 Non-Default 225694 88.38718

# Generate a pie chart to visualize the distribution of loan default status  
ggplot(data = default\_df, aes(x = '', y = Percent, fill = Status)) +  
 geom\_bar(stat = 'identity', width = 1) +  
 coord\_polar(theta = 'y') +  
 geom\_text(aes(label = sprintf('%.1f%%', Percent)), position = position\_stack(vjust = 0.6)) +  
 theme\_void() +   
 labs(title = 'Figure 3: Distribution of Loan Status')

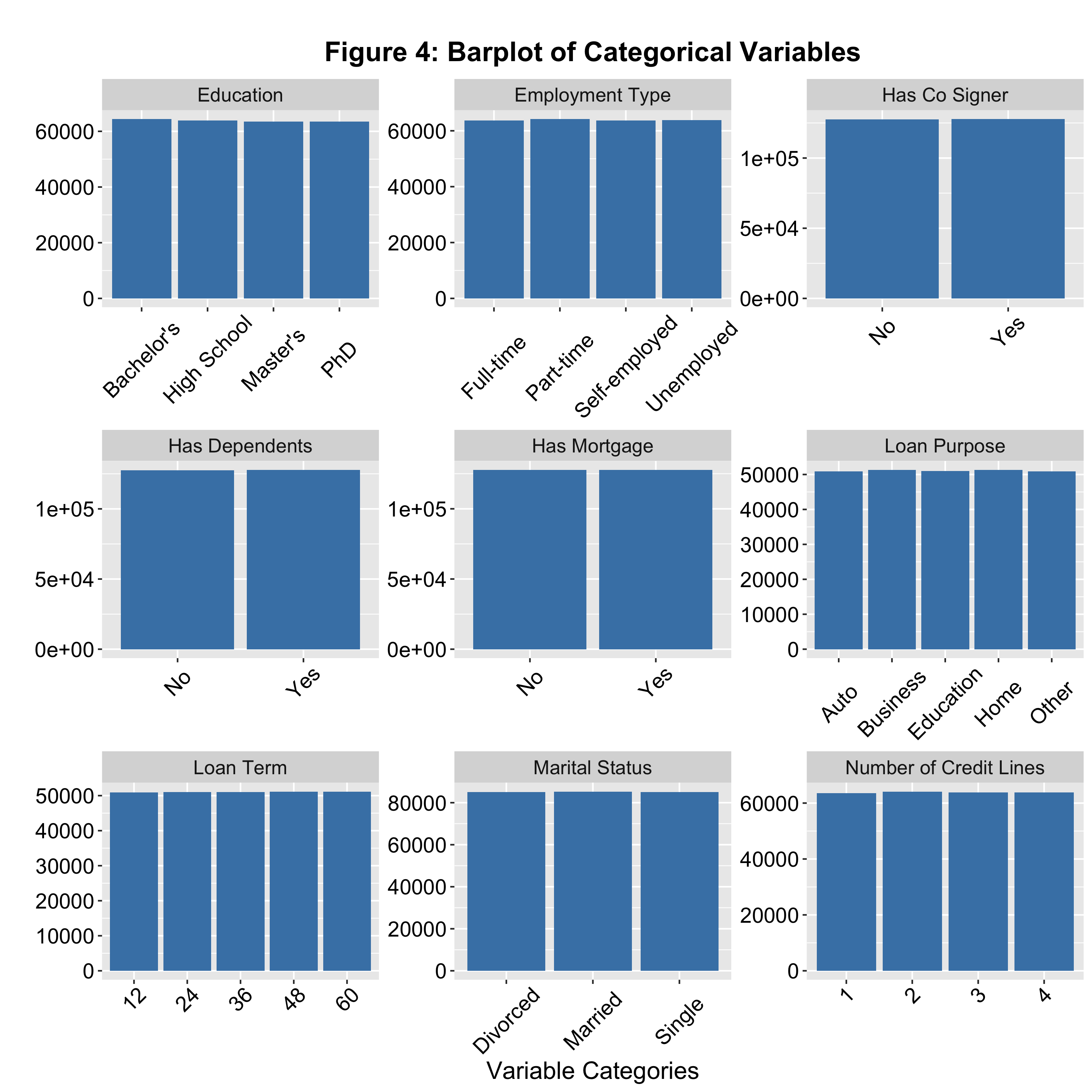


## Distribution of Categorical and Ordinal Variables

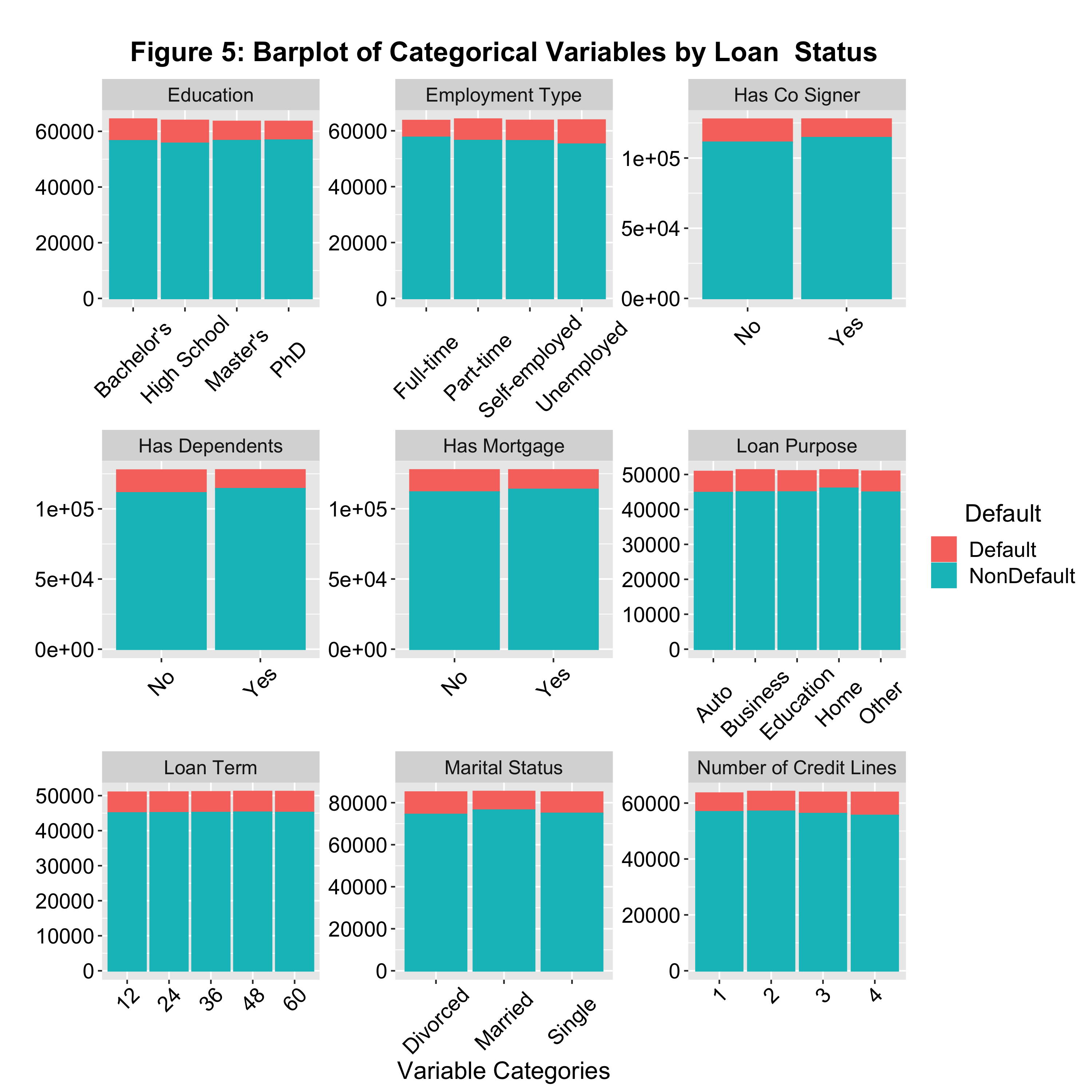
# Generate summary statistics for categorical and ordinal variables   
summary(df[c(categorical\_vars, ordinal\_vars)])

## Education EmploymentType MaritalStatus HasMortgage   
## Bachelor's :64366 Full-time :63656 Divorced:85033 No :127670   
## High School:63903 Part-time :64161 Married :85302 Yes:127677   
## Master's :63541 Self-employed:63706 Single :85012   
## PhD :63537 Unemployed :63824   
##   
## HasDependents LoanPurpose HasCoSigner NumCreditLines LoanTerm   
## No :127605 Auto :50844 No :127646 1:63554 12:50957   
## Yes:127742 Business :51298 Yes:127701 2:64130 24:51009   
## Education:51005 3:63834 36:51061   
## Home :51286 4:63829 48:51166   
## Other :50914 60:51154

# Reshape the dataframe to long format for plotting of categorical and ordinal variables  
df\_long\_cat = df[c(categorical\_vars, ordinal\_vars, target)] %>%  
 pivot\_longer(cols = -Default, names\_to = 'variable', values\_to = 'value')  
  
# Plot bar charts for each categorical and ordinal variable  
ggplot(data = df\_long\_cat) +  
 geom\_bar(aes(x = value), fill = 'steelblue') +  
 labs(title = 'Figure 4: Barplot of Categorical Variables', x = 'Variable Categories', y = '') +  
 facet\_wrap( ~ variable, scales = 'free', labeller = labeller(variable = var\_labels)) +   
 theme(axis.text.x = element\_text(angle = 45, hjust = 0.5))

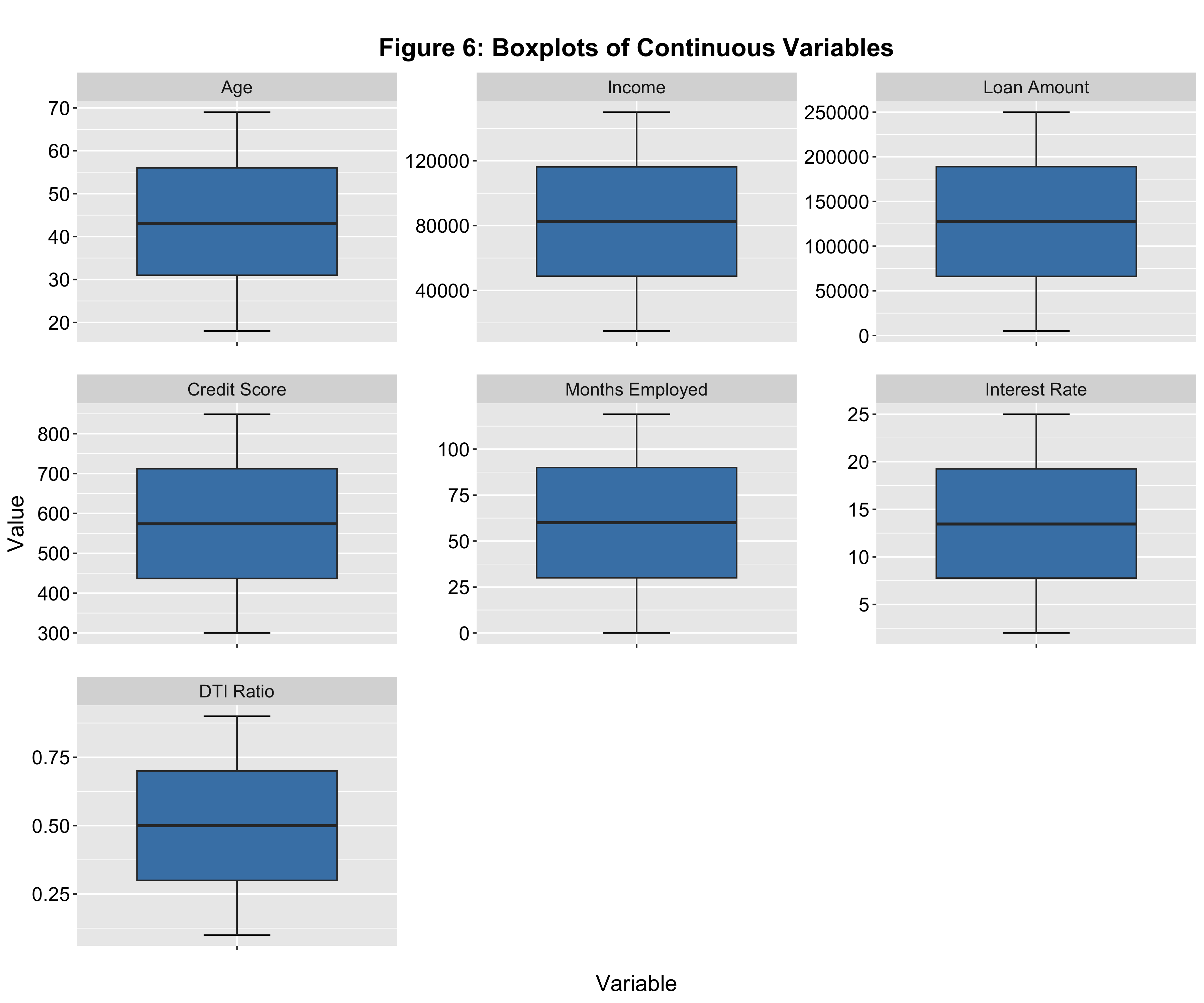


# Plot bar charts for each categorical and ordinal variable grouped by loan default status  
ggplot(data = df\_long\_cat) +  
 geom\_bar(aes(x = value, fill = Default, color = Default)) +  
 labs(title = 'Figure 5: Barplot of Categorical Variables by Loan Status', x = 'Variable Categories', y = '') +  
 facet\_wrap( ~ variable, scales = 'free', labeller = labeller(variable = var\_labels)) +   
 theme(axis.text.x = element\_text(angle = 45, hjust = 0.5))

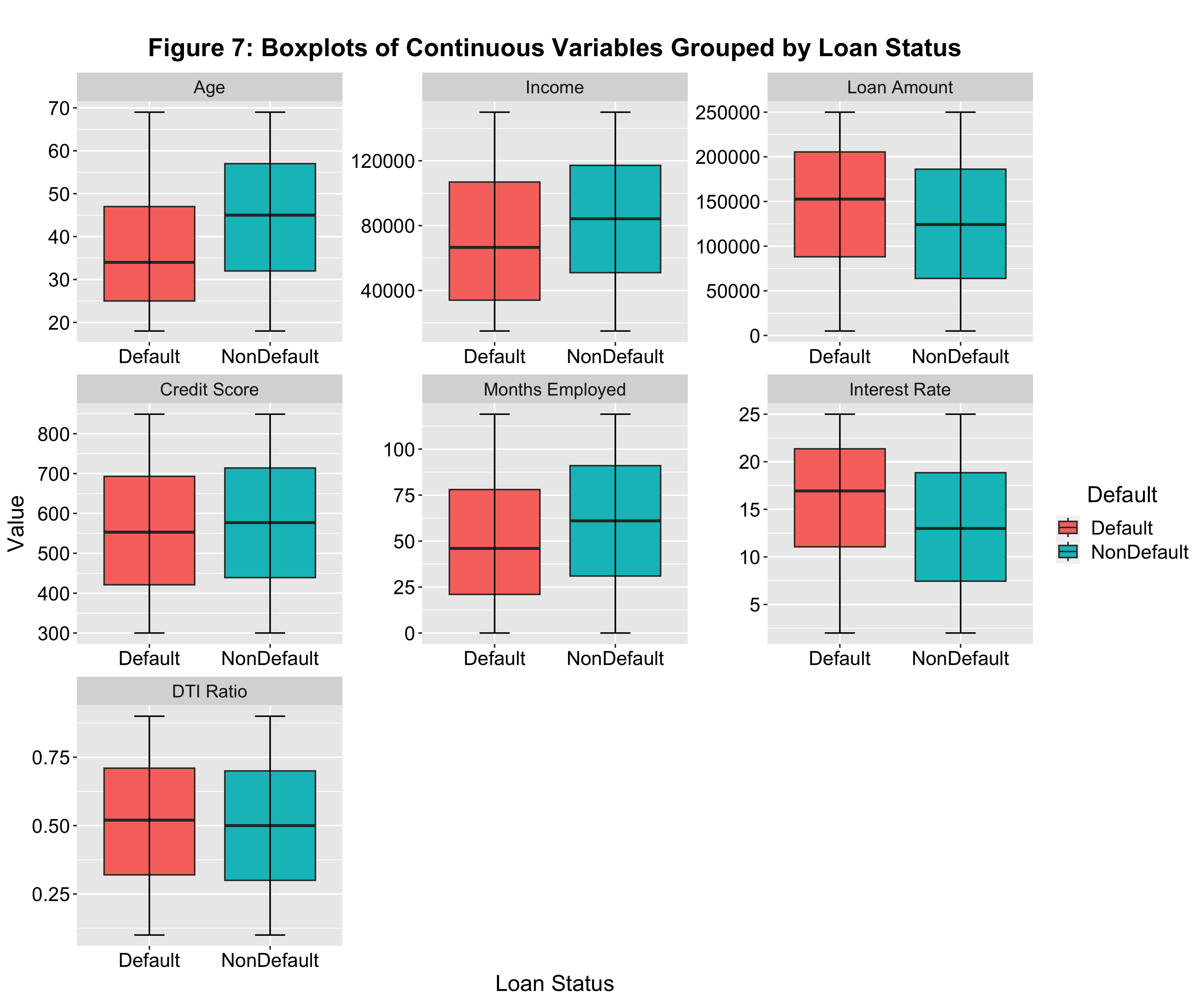


## Outlier Detection

# Plot a boxplot for each continuous variable to detect outliers  
ggplot(df\_long, aes(x = '', y = value)) +  
 stat\_boxplot(geom = 'errorbar',  
 width = 0.25) +   
 geom\_boxplot(fill = 'steelblue') +  
 facet\_wrap(~ variable, scales = 'free',   
 labeller = labeller(variable = var\_labels), ncol = 3) +  
 labs(title = 'Figure 6: Boxplots of Continuous Variables',   
 x = 'Variable', y = 'Value')



# Plot a boxplot for each continuous variable grouped by loan default status to detect outliers  
ggplot(df\_long, aes(x = Default, y = value, fill = Default)) +  
 geom\_boxplot() +   
 stat\_boxplot(geom = 'errorbar',  
 width = 0.25) +   
 facet\_wrap(~ variable, scales = 'free', labeller = labeller(variable = var\_labels)) +   
 labs(title = 'Figure 7: Boxplots of Continuous Variables Grouped by Loan Status',   
 x = 'Loan Status', y = 'Value')



# Main Data Analysis

## Spliting the data into train/validation/test sets

# Encode ordinal variable properly as an ordered factor  
for (var in ordinal\_vars) {  
 df[[var]] = factor(df[[var]], ordered = TRUE)  
}  
  
# Define the feature matrix  
X = df[, -which(names(df) == target)]  
# Get the target variable  
y = df[[target]]  
  
# Split the dataset into two equal parts: 50% for initial training and 50% for further splitting into validation and test sets  
set.seed(123)  
trainIndex = createDataPartition(y, p = 0.5, list = FALSE, times = 1)  
X\_train = X[trainIndex, ]   
X\_temp = X[-trainIndex, ]  
y\_train = y[trainIndex]  
y\_temp = y[-trainIndex]  
df\_train = df[trainIndex, ]  
df\_temp = df[-trainIndex, ]  
  
# Split the temp data into 20% validation and 30% test sets  
set.seed(123)  
valIndex = createDataPartition(y\_temp, p = 0.4, list = FALSE, times = 1)  
X\_val = X\_temp[valIndex, ]  
X\_test = X\_temp[-valIndex, ]  
y\_val = y\_temp[valIndex]  
y\_test = y\_temp[-valIndex]  
df\_val = df\_temp[valIndex, ]   
df\_test = df\_temp[-valIndex, ]

## Model Fitting with Cross-Validated Grid Search and Down Sampling

# Define training control parameters for a cross-validated grid search  
train\_control = trainControl(  
 method = 'cv', # Use cross-validation  
 number = 5, # Number of folds in cross-validation  
 classProbs = TRUE, # Enable computation of class probabilities  
 sampling = 'down', # Apply downsampling to address class imbalance  
 savePredictions = 'final', # Save predictions for the final model  
 summaryFunction = prSummary, # Use precision-recall summary function  
 allowParallel = TRUE # Allow parallel processing  
)  
  
# Fit a logistic regression model using glmnet (elastic net regularization)  
set.seed(123) # Set seed for reproducibility  
lr\_grid\_search = train(  
 Default ~ ., # Target variable Default predicted from all other variables  
 data = df\_train, # Training data  
 method = 'glmnet', # Logistic Regression model with built-in feature selection  
 trControl = train\_control, # Use defined training control settings  
 metric = 'AUC', # Optimization metric is the Area Under the Precision-Recall Curve  
 tuneLength = 5 # Number of different parameter combinations  
)  
  
# Fit a Random Forest model using ranger implementation  
set.seed(123) # Set seed for reproducibility  
rf\_grid\_search = train(  
 Default ~ ., # Target variable Default predicted from all other variables  
 data = df\_train, # Training data  
 method = 'ranger', # Random Forest algorithm using ranger package  
 trControl = train\_control, # Use defined training control settings  
 metric = 'AUC', # Optimization metric is the Area Under the Precision-Recall Curve  
 importance = 'impurity', # Feature importance based on impurity reduction  
 tuneLength = 5 # Number of different parameter combinations to try  
)  
  
# Fit an XGBoost model  
xgb\_grid\_search = train(  
 Default ~ ., # Target variable Default predicted from all other variables  
 data = df\_train, # Training data  
 method = 'xgbTree', # XGBoost model using xgbTree package  
 trControl = train\_control, # Use defined training control settings  
 metric = 'AUC', # Optimization metric is the Area Under the Precision-Recall Curve  
 tuneLength = 5 # Number of different parameter combinations to try  
)

## Model Threshold Tuning and Validation

# Define a function to calculate the optimal threshold using precision-recall curve  
calculateOptimalThreshold = function(y\_pred\_prob, df\_val) {  
 # Calculate the precision-recall curve for predicted probabilities and actual default statuses  
 pr\_curve\_obj = pr.curve(scores.class0 = y\_pred\_prob,   
 weights.class0 = df\_val$Default == 'Default',   
 curve = TRUE)  
 pr\_curve = pr\_curve\_obj$curve # Extract the precision-recall curve data  
   
 # Extract precision, recall, and threshold values from the curve  
 precisions = pr\_curve[ , 1]  
 recalls = pr\_curve[ , 2]  
 thresholds = pr\_curve[ , 3]  
 # Calculate the area under the precision-recall curve  
 auc = pr\_curve\_obj$auc.integral  
   
 # Compute F1 scores for each threshold and identify the index of the maximum F1 score  
 f1\_scores = 2 \* (precisions \* recalls) / (precisions + recalls)  
 optimal\_index = which.max(f1\_scores)  
 optimal\_threshold = thresholds[optimal\_index]  
 optimal\_precision = precisions[optimal\_index]  
 optimal\_recall = recalls[optimal\_index]  
 optimal\_f1\_score = f1\_scores[optimal\_index]  
   
 # Classify predictions based on the optimal threshold  
 y\_pred = factor(ifelse(y\_pred\_prob > optimal\_threshold, 'Default', 'NonDefault'),   
 levels = c('Default', 'NonDefault'))  
 y\_val = df\_val$Default  
   
 # Generate a confusion matrix from predictions  
 cm = table(Predicted = y\_pred, Actual = y\_val)  
   
 # Output the confusion matrix and performance metrics for the validation set  
 print('Performance on Validation Set:')  
 print(cm)  
   
 # Calculate accuracy from the confusion matrix  
 optimal\_accuracy = (cm['Default', 'Default'] +   
 cm['NonDefault', 'NonDefault']) / sum(cm)  
   
 # Store and print optimal evaluation metrics  
 eval\_metrics = c('Accuracy' = optimal\_accuracy, 'Precision' = optimal\_precision,   
 'Recall' = optimal\_recall, 'AUC' = auc,   
 'F1-Score' = optimal\_f1\_score, 'Thresold' = optimal\_threshold)  
 print('Optimal Metrics:')  
 print(eval\_metrics)  
   
 # Return the optimal threshold  
 return(optimal\_threshold)  
}  
  
# Logistic Regression:  
# Get the predicted probabilities of default across the validation set  
y\_val\_pred\_prob = predict(lr\_grid\_search, newdata = X\_val, type = 'prob')[ , 'Default']  
# Calculate the optimal threshold for via AUPRC  
optimal\_threshold\_lr = calculateOptimalThreshold(y\_val\_pred\_prob, df\_val)

## [1] "Performance on Validation Set:"  
## Actual  
## Predicted Default NonDefault  
## Default 2947 7540  
## NonDefault 2984 37599  
## [1] "Optimal Metrics:"  
## Accuracy Precision Recall AUC F1-Score Thresold   
## 0.7939299 0.4970494 0.2810831 0.2983166 0.3590962 0.6143340

# Random Forest:  
# Get the predicted probabilities of default across the validation set  
y\_val\_pred\_prob = predict(rf\_grid\_search, newdata = X\_val, type = 'prob')[ , 'Default']  
# Calculate the optimal threshold for via AUPRC  
optimal\_threshold\_rf = calculateOptimalThreshold(y\_val\_pred\_prob, df\_val)

## [1] "Performance on Validation Set:"  
## Actual  
## Predicted Default NonDefault  
## Default 2811 7111  
## NonDefault 3120 38028  
## [1] "Optimal Metrics:"  
## Accuracy Precision Recall AUC F1-Score Thresold   
## 0.7996671 0.4739504 0.2833098 0.2908384 0.3546332 0.5920000

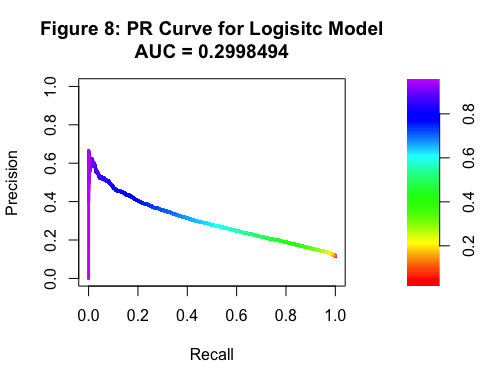
# XGBoost:  
# Get the predicted probabilities of default across the validation set  
y\_val\_pred\_prob = predict(xgb\_grid\_search, newdata = X\_val, type = 'prob')[ , 'Default']  
# Calculate the optimal threshold for via AUPRC  
optimal\_threshold\_xgb = calculateOptimalThreshold(y\_val\_pred\_prob, df\_val)

## [1] "Performance on Validation Set:"  
## Actual  
## Predicted Default NonDefault  
## Default 2920 7298  
## NonDefault 3011 37841  
## [1] "Optimal Metrics:"  
## Accuracy Precision Recall AUC F1-Score Thresold   
## 0.7981398 0.4924970 0.2858401 0.3090570 0.3617337 0.6228722

## Model Evaluation

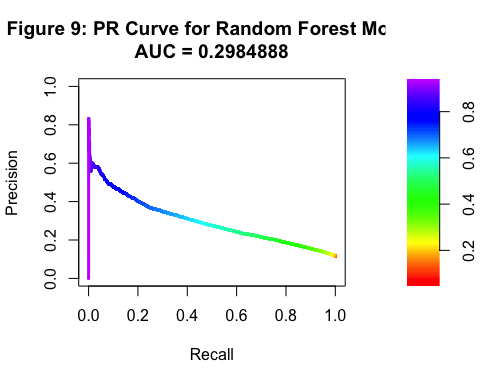
### Performce Metrics

# Define a function to evaluate model performance based on predicted probabilities, an optimal threshold, test data, and a plot title  
evaluatePerformance = function(y\_pred\_prob, optimal\_threshold, df\_test, title) {  
 # Calculate the precision-recall curve for predicted probabilities and actual default status, then plot it  
 pr\_curve\_obj = pr.curve(scores.class0 = y\_pred\_prob,   
 weights.class0 = df\_test$Default == 'Default', curve = TRUE)  
 plot(pr\_curve\_obj, main = title)  
   
 # Extract the area under the precision-recall curve (AUC)  
 auc = pr\_curve\_obj$auc.integral  
   
 # Prepare actual and predicted responses based on the optimal threshold  
 y\_test = df\_test$Default  
 y\_pred = factor(ifelse(y\_pred\_prob > optimal\_threshold, 'Default', 'NonDefault'),   
 levels = c('Default', 'NonDefault'))  
   
 # Generate and output a confusion matrix to evaluate classification accuracy  
 cm = table(Predicted = y\_pred, Actual = y\_test)  
 print('Performance on Test Set:')  
 print(cm)  
   
 # Calculate accuracy, precision, recall, and F1 score from the confusion matrix  
 accuracy = (cm['Default', 'Default'] + cm['NonDefault', 'NonDefault']) / sum(cm)  
 precision = cm['Default', 'Default'] / sum(cm['Default', ])  
 recall = cm['Default', 'Default'] / sum(cm[, 'Default'])  
 f1\_score = 2 \* (precision \* recall) / (precision + recall)  
   
 # Store, output, and return all evaluation metrics on the test set  
 eval\_metrics = c('Accuracy' = accuracy, 'Precision' = precision,  
 'Recall' = recall, 'AUC' = auc,  
 'F1-Score' = f1\_score, 'Thresold' = optimal\_threshold)  
 print('Evaluation Metrics:')  
 print(eval\_metrics)  
 return(eval\_metrics)  
}  
  
# Evaluate Logistic Regression model performance on the test dataset and plot the PR curve  
y\_test\_pred\_prob = predict(lr\_grid\_search, newdata = X\_test, type = 'prob')[ , 'Default']  
perf\_results\_lr = evaluatePerformance(y\_test\_pred\_prob, optimal\_threshold\_lr, df\_test, title = 'Figure 8: PR Curve for Logisitc Model')



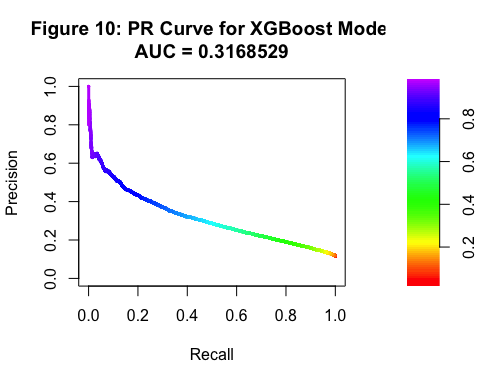
## [1] "Performance on Test Set:"  
## Actual  
## Predicted Default NonDefault  
## Default 4392 11234  
## NonDefault 4503 56474  
## [1] "Evaluation Metrics:"  
## Accuracy Precision Recall AUC F1-Score Thresold   
## 0.7945642 0.2810700 0.4937605 0.2998494 0.3582236 0.6143340

# Evaluate Random Forest model performance on the test dataset and plot the PR curve  
y\_test\_pred\_prob = predict(rf\_grid\_search, newdata = X\_test, type = 'prob')[ , 'Default']  
perf\_results\_rf = evaluatePerformance(y\_test\_pred\_prob, optimal\_threshold\_rf, df\_test, title = 'Figure 9: PR Curve for Random Forest Model')



## [1] "Performance on Test Set:"  
## Actual  
## Predicted Default NonDefault  
## Default 4243 10760  
## NonDefault 4652 56948  
## [1] "Evaluation Metrics:"  
## Accuracy Precision Recall AUC F1-Score Thresold   
## 0.7988068 0.2828101 0.4770096 0.2984888 0.3550925 0.5920000

# Evaluate XGBoost model performance on the test dataset and plot the PR curve  
y\_test\_pred\_prob = predict(xgb\_grid\_search, newdata = X\_test, type = 'prob')[ , 'Default']  
perf\_results\_xgb = evaluatePerformance(y\_test\_pred\_prob, optimal\_threshold\_xgb, df\_test, title = 'Figure 10: PR Curve for XGBoost Model')



## [1] "Performance on Test Set:"  
## Actual  
## Predicted Default NonDefault  
## Default 4406 10911  
## NonDefault 4489 56797  
## [1] "Evaluation Metrics:"  
## Accuracy Precision Recall AUC F1-Score Thresold   
## 0.7989635 0.2876542 0.4953345 0.3168529 0.3639518 0.6228722

# Combine and output performance results from all models on the test set  
perf\_results = rbind(perf\_results\_lr, perf\_results\_rf, perf\_results\_xgb)  
rownames(perf\_results) = c('Logistic Regression', 'Random Forest', 'XGBoost')  
perf\_results

## Accuracy Precision Recall AUC F1-Score Thresold  
## Logistic Regression 0.7945642 0.2810700 0.4937605 0.2998494 0.3582236 0.6143340  
## Random Forest 0.7988068 0.2828101 0.4770096 0.2984888 0.3550925 0.5920000  
## XGBoost 0.7989635 0.2876542 0.4953345 0.3168529 0.3639518 0.6228722

### Variable Importance

# Retrieve the final model from the XGBoost grid search results  
best\_xgboost\_model = xgb\_grid\_search$finalModel  
# Compute variable importance scores from the XGBoost model without scaling the importance values  
var\_importance = varImp(xgb\_grid\_search, scale = FALSE)  
  
# Create a dataframe for plotting, extracting variable names and their corresponding importance scores  
importance\_df = data.frame(Variable = rownames(var\_importance$importance),  
 Importance = var\_importance$importance$Overall)  
  
# Generate a bar plot of the variable importances from the XGBoost model  
ggplot(importance\_df, aes(x = reorder(Variable, Importance), y = Importance)) +  
 geom\_bar(stat = 'identity', fill = 'steelblue') +  
 coord\_flip() +  
 labs(title = 'Figure 11: XGBoost Variable Importance Plot',  
 x = 'Variable', y = 'Importance') +  
 theme(plot.title = element\_text(hjust = 0.5))

