Assignement- 2

Ahaan Tagare

2025-01-13

Student - 33865799

STATISTICS AND STATISTICAL DATA MINING

Predictive Modelling for Loan Approval: A Comparative Analysis of Machine Learning Techniques

Dataset - https://www.kaggle.com/datasets/zaurbegiev/my-dataset?select=credit_train.csv

The code's libraries perform a number of vital operations for data analysis and machine learning projects While dplyr and tidyr are used for data manipulation and tidying, R. gbm is used for gradient boosting models. Caret provides a uniform interface for training and assessing machine learning models, while ggplot2 makes sophisticated visualizations possible. ROC curves and AUC are used by pROC to assess classifier performance, while randomforest creates random forest models. rpart and rpart.plot with an emphasis on decision tree visualizations and models. PDP uses partial dependence plots to help visualize model behavior, and Rumble offers a graphical user interface for data science. NeuralNetTools helps with neural network analysis, nnet is used for neural network training, and coefplot visualizes regression model coefficients. Data preparation and imbalanced datasets are handled by DMwR, while DiagrammeR generates graphical diagrams for network or workflow visualization. Finally, data can be reshaped between wide and long formats using reshape2. Together, these packages offer a complete toolkit for modeling, data wrangling, visualization, and model validation, have suppressed the warnings because of the setting control in the PC in this code which would make no impact on the model.

```r
suppressPackageStartupMessages({
  suppressWarnings({
    library(gbm)
    library(dplyr)
    library(tidyr)
    library(ggplot2)
    library(caret)
    library(randomForest)
    library(pROC)
    library(rpart)
    library(rpart.plot)
    library(rattle)
    library(pdp)
    library(coefplot)
    library(NeuralNetTools)
    library(nnet)
    library(DMwR)
    library(DiagrammeR)
    library(reshape2)
    library(scales)
```

```
  })
})
```

```
# Data preprocessing
data <- read.csv("Credit_data.csv")
str(data)
```

```
## 'data.frame':    100514 obs. of  19 variables:
##  $ Loan.ID                     : chr  "14dd8831-6af5-400b-83ec-68e61888a048" "4771cc26-131a-45db-b5aa
##  $ Customer.ID                 : chr  "981165ec-3274-42f5-a3b4-d104041a9ca9" "2de017a3-2e01-49cb-a581
##  $ Loan.Status                 : chr  "Fully Paid" "Fully Paid" "Fully Paid" "Fully Paid" ...
##  $ Current.Loan.Amount         : int  445412 262328 99999999 347666 176220 206602 217646 648714 54874
##  $ Term                        : chr  "Short Term" "Short Term" "Short Term" "Long Term" ...
##  $ Credit.Score                : int  709 NA 741 721 NA 7290 730 NA 678 739 ...
##  $ Annual.Income               : int  1167493 NA 2231892 806949 NA 896857 1184194 NA 2559110 1454735
##  $ Years.in.current.job        : chr  "8 years" "10+ years" "8 years" "3 years" ...
##  $ Home.Ownership              : chr  "Home Mortgage" "Home Mortgage" "Own Home" "Own Home" ...
##  $ Purpose                     : chr  "Home Improvements" "Debt Consolidation" "Debt Consolidation"
##  $ Monthly.Debt                : num  5215 33296 29201 8742 20640 ...
##  $ Years.of.Credit.History     : num  17.2 21.1 14.9 12 6.1 17.3 19.6 8.2 22.6 13.9 ...
##  $ Months.since.last.delinquent: int  NA 8 29 NA NA NA 10 8 33 NA ...
##  $ Number.of.Open.Accounts     : int  6 35 18 9 15 6 13 15 4 20 ...
##  $ Number.of.Credit.Problems   : int  1 0 1 0 0 0 1 0 0 0 ...
##  $ Current.Credit.Balance      : int  228190 229976 297996 256329 253460 215308 122170 193306 437171
##  $ Maximum.Open.Credit         : int  416746 850784 750090 386958 427174 272448 272052 864204 555038
##  $ Bankruptcies                : int  1 0 0 0 0 0 1 0 0 0 ...
##  $ Tax.Liens                   : int  0 0 0 0 0 0 0 0 0 0 ...
```

```
# Create the flowchart
flowchart <- grViz("
digraph workflow {
  graph [layout = dot, rankdir = TB]

  # Nodes
  node [shape = box, style = filled, color = lightblue]
  A [label = 'Load and Inspect Data']
  B [label = 'Data Cleaning and Preprocessing']
  C1 [label = 'Handle Missing Values']
  C2 [label = 'Variable Conversion']
  D [label = 'Feature Engineering']
  D1 [label = 'Calculate DTI']
  D2 [label = 'Credit Utilization']
  E [label = 'Train-Test Split']
  F [label = 'Model Training and Evaluation']
  F1 [label = 'Decision Tree']
  F2 [label = 'Random Forest']
  F3 [label = 'Logistic Regression']
  F4 [label = 'Gradient Boosting']
  F5 [label = 'Artificial Neural Network']
  G [label = 'Model Evaluation']
  G1 [label = 'Accuracy, Precision, Recall, F1 Score']
  H [label = 'Model Comparison']
  I [label = 'ROC Analysis']

  # Edges
```

```
  A -> B
  B -> C1
  B -> C2
  C1 -> D
  C2 -> D
  D -> D1
  D -> D2
  D1 -> E
  D2 -> E
  E -> F
  F -> F1
  F -> F2
  F -> F3
  F -> F4
  F -> F5
  F1 -> G
  F2 -> G
  F3 -> G
  F4 -> G
  F5 -> G
  G -> G1
  G1 -> H
  H -> I
}
")

# Render the flowchart
flowchart
```

```
## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is installed, please
```

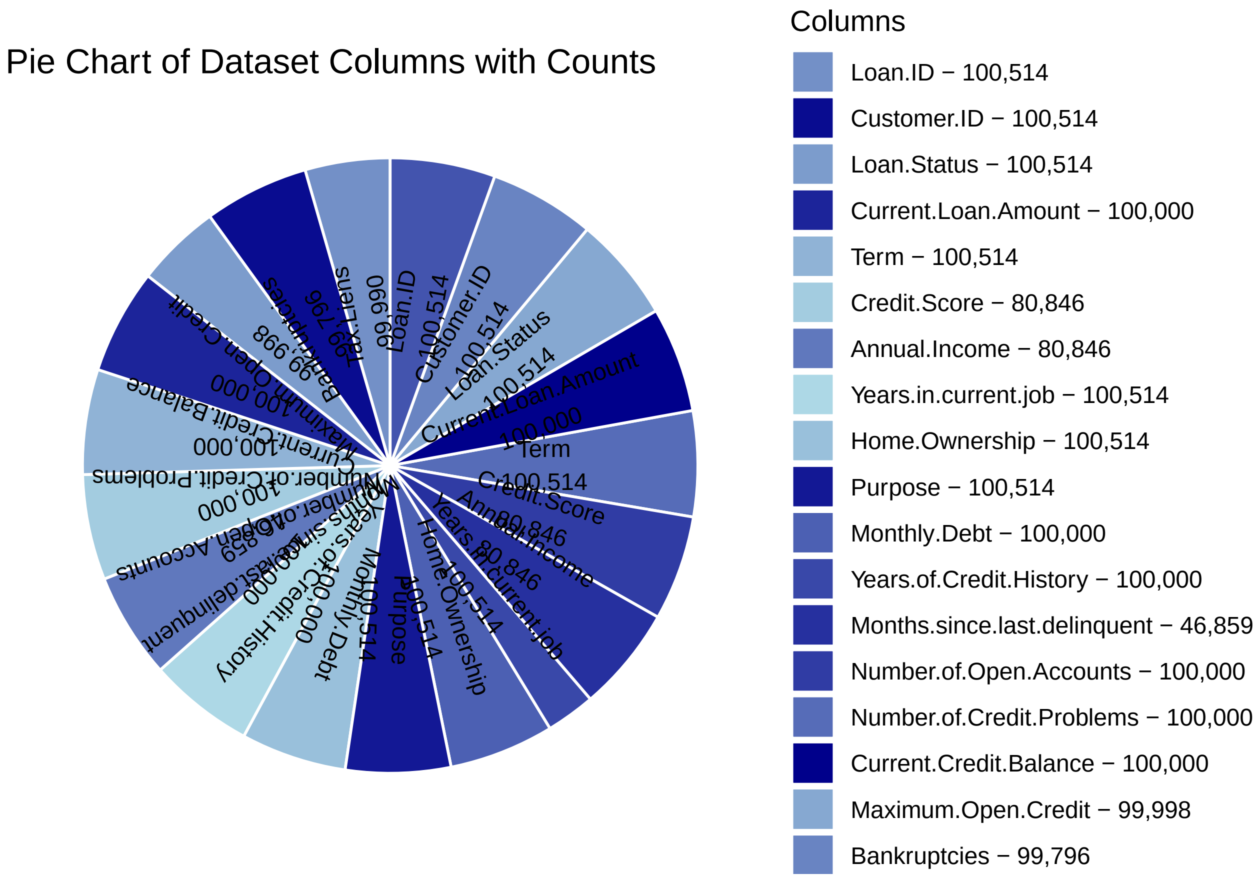
This code prepares data for a pie chart that displays column counts by loading a dataset and counting non-missing values for each column. After determining the cumulative midpoints and proportions for label placement, it gives each column a distinct color. The text labels are positioned inside the chart slices after a pie chart with personalized labels, colors, and a legend is created using ggplot2. As a result, column names and counts are shown in an eye-catching pie chart.

```
# Data preprocessing
data <- read.csv("Credit_data.csv")
str(data)
```

```
## 'data.frame':    100514 obs. of  19 variables:
##  $ Loan.ID                     : chr  "14dd8831-6af5-400b-83ec-68e61888a048" "4771cc26-131a-45db-b5a
##  $ Customer.ID                 : chr  "981165ec-3274-42f5-a3b4-d104041a9ca9" "2de017a3-2e01-49cb-a58
##  $ Loan.Status                 : chr  "Fully Paid" "Fully Paid" "Fully Paid" "Fully Paid" ...
##  $ Current.Loan.Amount         : int  445412 262328 99999999 347666 176220 206602 217646 648714 5487
##  $ Term                        : chr  "Short Term" "Short Term" "Short Term" "Long Term" ...
##  $ Credit.Score                : int  709 NA 741 721 NA 7290 730 NA 678 739 ...
##  $ Annual.Income               : int  1167493 NA 2231892 806949 NA 896857 1184194 NA 2559110 1454735
##  $ Years.in.current.job        : chr  "8 years" "10+ years" "8 years" "3 years" ...
##  $ Home.Ownership              : chr  "Home Mortgage" "Home Mortgage" "Own Home" "Own Home" ...
##  $ Purpose                     : chr  "Home Improvements" "Debt Consolidation" "Debt Consolidation"
##  $ Monthly.Debt                : num  5215 33296 29201 8742 20640 ...
##  $ Years.of.Credit.History     : num  17.2 21.1 14.9 12 6.1 17.3 19.6 8.2 22.6 13.9 ...
##  $ Months.since.last.delinquent: int  NA 8 29 NA NA NA 10 8 33 NA ...
```

```
##  $ Number.of.Open.Accounts     : int  6 35 18 9 15 6 13 15 4 20 ...
##  $ Number.of.Credit.Problems   : int  1 0 1 0 0 0 1 0 0 0 ...
##  $ Current.Credit.Balance      : int  228190 229976 297996 256329 253460 215308 122170 193306 437171
##  $ Maximum.Open.Credit         : int  416746 850784 750090 386958 427174 272448 272052 864204 555038
##  $ Bankruptcies                : int  1 0 0 0 0 0 1 0 0 0 ...
##  $ Tax.Liens                   : int  0 0 0 0 0 0 0 0 0 0 ...
```

```r
column_names <- colnames(data)
counts <- colSums(!is.na(data))

# Prepare the data forpie chart
pie_data <- data.frame(Column = column_names, Count = counts)


pie_data$Proportion <- pie_data$Count / sum(pie_data$Count)
pie_data$Cumulative <- cumsum(pie_data$Proportion) - pie_data$Proportion / 2


pie_data$Legend_Label <- paste0(pie_data$Column, " - ", scales::comma(pie_data$Count))


color_palette <- colorRampPalette(c("lightblue", "darkblue"))(nrow(pie_data))



ggplot(pie_data, aes(x = "", y = Count, fill = Legend_Label)) +
  geom_bar(stat = "identity", width = 1, color = "white") +
  coord_polar("y", start = 0) +
  theme_void() +
  labs(
    title = "Pie Chart of Dataset Columns with Counts",
    fill = "Columns"
  ) +
  theme(legend.position = "right") +
  scale_fill_manual(
    values = setNames(color_palette, pie_data$Legend_Label),  # Match colors explicitly to labels
    labels = pie_data$Legend_Label
  ) +
  geom_text(
    aes(
      y = Cumulative * sum(Count),
      label = paste(Column, scales::comma(Count), sep = "\n")  # Add names and counts
    ),
    size = 3,
    angle = 90 - (pie_data$Cumulative * 360),
    hjust = 0.5, color = "black"
  )
```



This code preprocesses the data by substituting 0 for any missing values in the dataset. The "Short Term" value is changed to 0 and other values to 1, converting the Term column to binary values. Additionally, it uses the mean of the current non-missing values to fill in the missing values in the Months.since.last.delinquent column. These changes get the data ready for additional modeling or analysis.

```
data[is.na(data)] <- 0
data$Term <- ifelse(data$Term == "Short Term", 0, 1)
data$Months.since.last.delinquent[is.na
(data$Months.since.last.delinquent)] <- mean(data$Months.since.last.delinquent, na.rm = TRUE)
```

This code cleans the `Years.in.current.job` column by removing the word years and replacing 10+ with 10. It converts the column to numeric, suppressing warnings for non-numeric values. Missing or invalid values are replaced with 0. Finally, it prints the dimensions of the cleaned data set using `dim(data)`.

```
# Clean Years

data$Years.in.current.job <- gsub(" years", "", data$Years.in.current.job)
data$Years.in.current.job[data$Years.in.current.job == "10+"] <- 10
data$Years.in.current.job <- suppressWarnings(as.numeric(data$Years.in.current.job))
data$Years.in.current.job[is.na(data$Years.in.current.job)] <- 0

print(dim(data))
```

```
## [1] 100514     19
```

The Home.Ownership and Purpose columns are transformed into factors by this code. For both columns, it retrieves the levels (categories) and the numeric codes that correspond to them. It generates a table for each that associates the levels with their respective numeric codes (purpose_table for Purpose and home_ownership_table for Home.Ownership). The mapping of factor levels to corresponding codes is then shown by printing these tables.

```r
# Converting the columns to factors
data$Home.Ownership <- as.factor(data$Home.Ownership)
data$Purpose <- as.factor(data$Purpose)


home_ownership_levels <- levels(data$Home.Ownership)
home_ownership_codes <- as.integer(data$Home.Ownership)


purpose_levels <- levels(data$Purpose)
purpose_codes <- as.integer(data$Purpose)


home_ownership_table <- data.frame(Level = home_ownership_levels,
Code = unique(home_ownership_codes))


purpose_table <- data.frame(Level = purpose_levels,
Code = unique(purpose_codes))


print(home_ownership_table)
```

```
##           Level Code
## 1                  3
## 2  HaveMortgage    4
## 3 Home Mortgage    5
## 4      Own Home    2
## 5          Rent    1
```

```r
print(purpose_table)
```

```
##                   Level Code
## 1                          7
## 2         Business Loan    5
## 3             Buy a Car    4
## 4             Buy House   11
## 5    Debt Consolidation    2
## 6  Educational Expenses    3
## 7     Home Improvements    8
## 8        major_purchase   15
## 9         Medical Bills   12
## 10               moving   14
## 11                other    9
## 12                Other   17
## 13     renewable_energy   16
## 14       small_business    6
## 15          Take a Trip   10
## 16             vacation   13
## 17              wedding    1
```

This code performs feature engineering by creating two new features: `DTI` (Debt-to-Income Ratio) and `Credit.Utilization`. The `DTI` is calculated as the ratio of `Monthly.Debt` to `Annual.Income`, and

`Credit.Utilization` is calculated as the ratio of `Current.Credit.Balance` to `Maximum.Open.Credit`. To ensure numerical stability, any missing or infinite values resulting from these calculations are replaced with 0. Finally, the first few values of both `DTI` and `Credit.Utilization` are printed using `head()` to verify the successful computation and handling of these features.

```
# Feature Engineering
data$DTI <- data$Monthly.Debt / data$Annual.Income
data$DTI[is.na(data$DTI) | is.infinite(data$DTI)] <- 0
data$Credit.Utilization <- data$Current.Credit.Balance /
data$Maximum.Open.Credit
data$Credit.Utilization[is.na(data$Credit.Utilization)] <- 0

print(head(data$DTI))
```

```
## [1] 0.004466614 0.000000000 0.013083308 0.010833274 0.000000000 0.018250111
```

```
print(head(data$Credit.Utilization))
```

```
## [1] 0.5475517 0.2703107 0.3972803 0.6624207 0.5933414 0.7902719
```

This code creates a new binary variable, `LoanApproval`, by assigning a value of `1` if `Current.Loan.Amount` exceeds 500,000 and `0` otherwise. After printing the dimensions of the dataset (`dim(data)`), it partitions the data set into training and testing subsets using a 70-30 split, controlled by setting a random seed (`set.seed(123)`) for reproducibility. The `createDataPartition()` function ensures that the distribution of `LoanApproval` is preserved in both subsets. Finally, it prints the dimensions of `train_data` and `test_data` to confirm the partitioning.

```
data$LoanApproval <- ifelse(data$Current.Loan.Amount > 500000, 1, 0)
print(dim(data))
```

```
## [1] 100514     22
```

```
trainIndex <- createDataPartition(data$LoanApproval, p = 0.7, list = FALSE)
train_data <- data[trainIndex, ]
test_data <- data[-trainIndex, ]

dim(train_data)
```

```
## [1] 70360    22
```

```
dim(test_data)
```

```
## [1] 30154    22
```

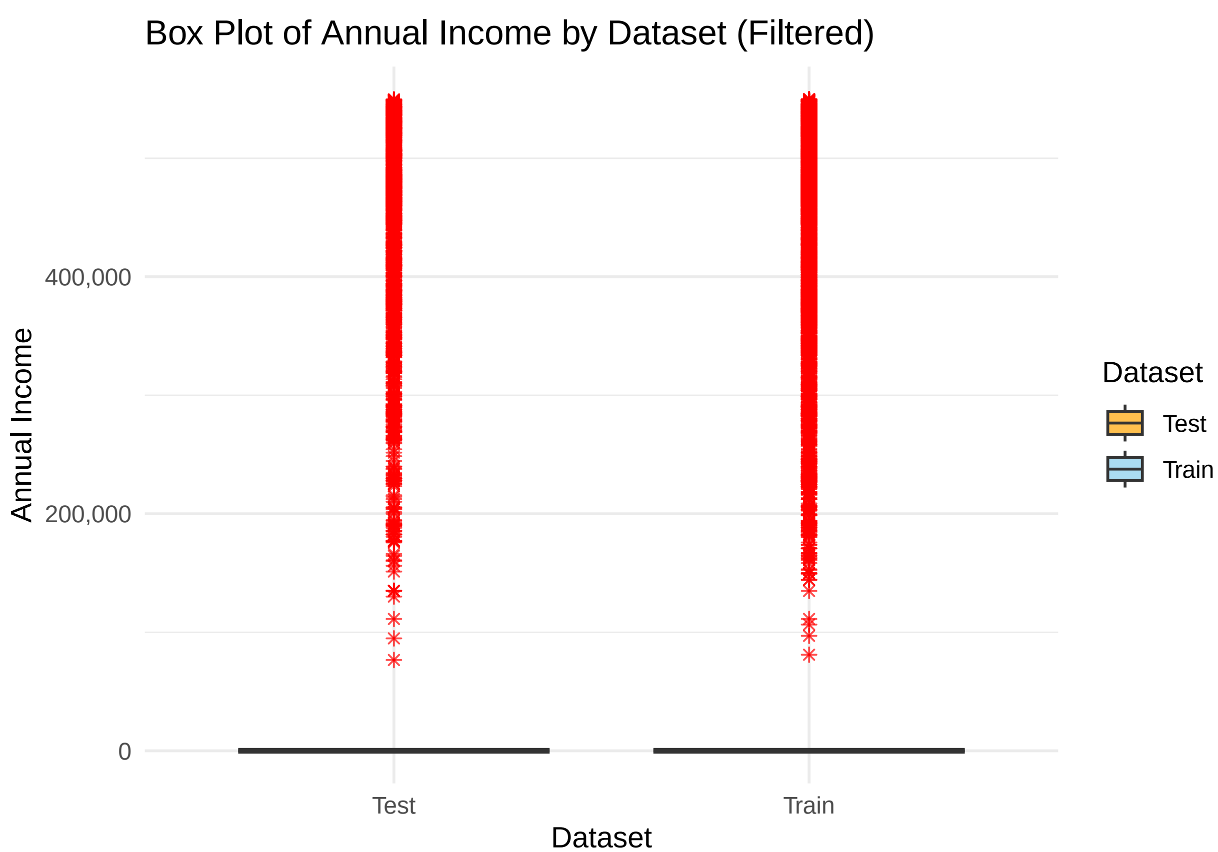
Finally, it prints the dimensions of `train_data` and `test_data` to confirm the partitioning.undefined`undefined`undefined`undefi`

```
train_data[sapply(train_data, is.infinite)] <- 0
test_data[sapply(test_data, is.infinite)] <- 0
```

The code first uses scale to standardize the numeric columns that are extracted from train_data. The standardized data (prcomp) is then subjected to Principal Component Analysis (PCA), and a summary of the PCA is then given. To see each principal component's explained variance, a scree plot is created. Head(pca_train_data) is used to display the pca_train_data, which is created by combining the target variable LoanApproval with the first few principal components that are kept in pca_data.

```
#Box plot
train_data$Dataset <- "Train"
test_data$Dataset <- "Test"
```

```r
# Combine
combined_data <- rbind(
  train_data[, c("Annual.Income", "Dataset")],
  test_data[, c("Annual.Income", "Dataset")]
)

# Filter
filtered_data <- combined_data[combined_data$Annual.Income < 550000, ]  # Keep values < 500,000

# box plot
ggplot(filtered_data, aes(x = Dataset, y = Annual.Income, fill = Dataset)) +
  geom_boxplot(outlier.color = "red", outlier.shape = 8, alpha = 0.7) +
  theme_minimal() +
  labs(
    title = "Box Plot of Annual Income by Dataset (Filtered)",
    x = "Dataset",
    y = "Annual Income"
  ) +
  scale_fill_manual(values = c("Train" = "skyblue", "Test" = "orange")) +
  theme(axis.text.x = element_text(angle = 0, hjust = 0.5)) +
  scale_y_continuous(labels = scales::comma)
```

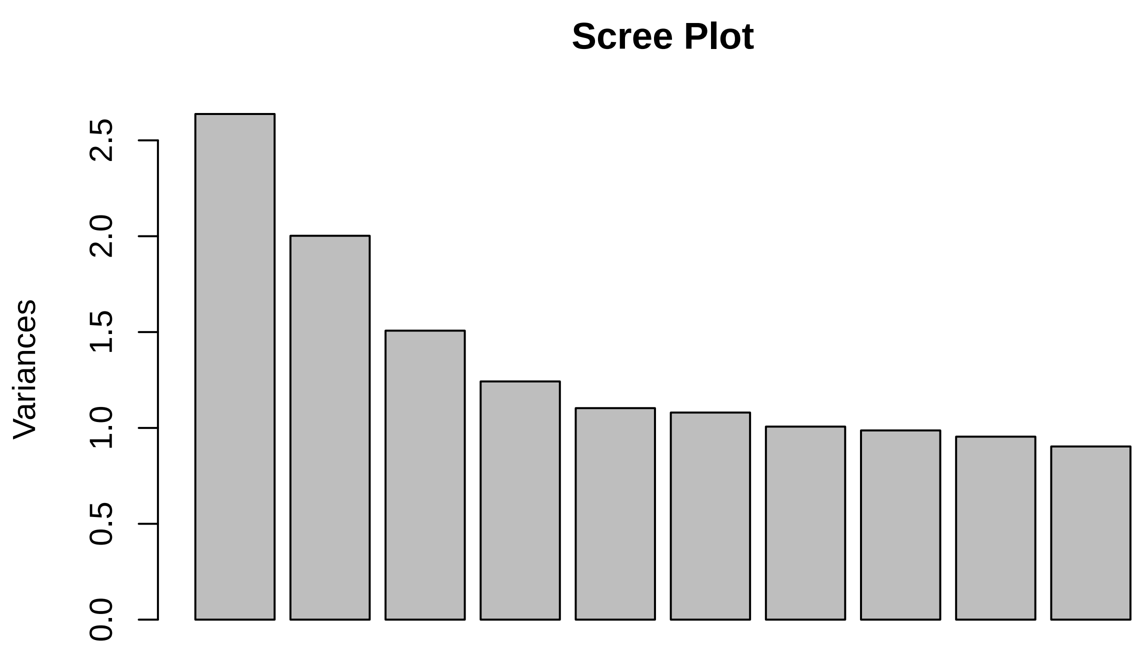


The code first uses scale to standardize the numeric columns that are extracted from train_data. The standardized data (prcomp) is then subjected to Principal Component Analysis (PCA), and a summary of the PCA is then given. To see each principal component's explained variance, a scree plot is created. Head(pca_train_data) is used to display the pca_train_data, which is created by combining the target variable LoanApproval with the first few principal components that are kept in pca_data.

```
train_data_numeric <- train_data[, sapply(train_data, is.numeric)]

# Standardize
train_data_scaled <- scale(train_data_numeric)

pca <- prcomp(train_data_scaled, center = TRUE, scale. = TRUE)

# Summary
summary(pca)
```

```
## Importance of components:
##                           PC1    PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     1.6240 1.4150 1.22772 1.11464 1.05033 1.03929 1.00344
## Proportion of Variance 0.1551 0.1178 0.08866 0.07308 0.06489 0.06354 0.05923
## Cumulative Proportion  0.1551 0.2729 0.36159 0.43467 0.49956 0.56310 0.62233
##                            PC8     PC9    PC10    PC11    PC12   PC13    PC14
## Standard deviation     0.99346 0.97694 0.95041 0.92061 0.89014 0.8153 0.71273
## Proportion of Variance 0.05806 0.05614 0.05313 0.04985 0.04661 0.0391 0.02988
## Cumulative Proportion  0.68039 0.73653 0.78966 0.83952 0.88613 0.9252 0.95511
##                          PC15    PC16    PC17
## Standard deviation     0.6387 0.53571 0.26117
## Proportion of Variance 0.0240 0.01688 0.00401
## Cumulative Proportion  0.9791 0.99599 1.00000
```

```
screeplot(pca, main = "Scree Plot")
```

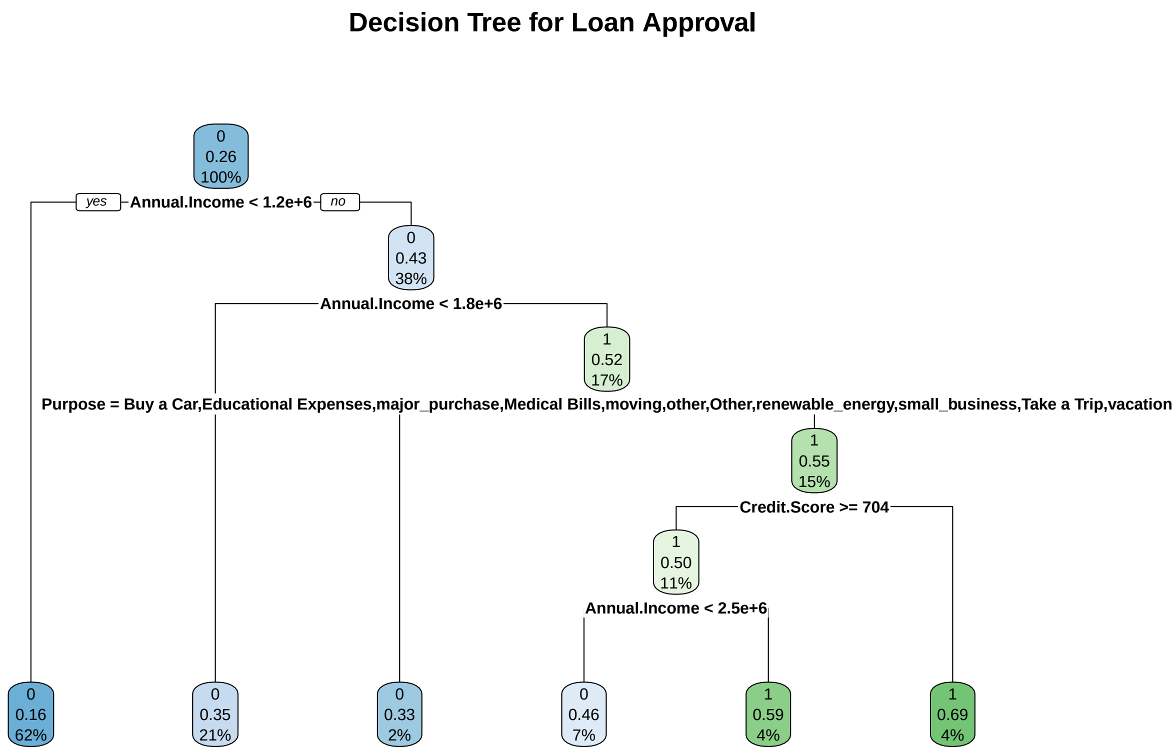


```r
pca_data <- data.frame(pca$x)


pca_train_data <- cbind(pca_data, LoanApproval = train_data$LoanApproval)

head(pca_train_data)
```

```
##           PC1         PC2       PC3        PC4        PC5        PC6        PC7
## 1  -1.9750531 -2.03772470 0.2351060  0.2759529  1.0054933  0.3990721 -0.1478831
## 2   1.0615279 -0.09163899 1.1039201  2.3641241 -2.6698199 -1.0219761  0.1540961
## 4  -0.6969005  1.02238371 0.4108680 -0.5493675  1.1432378 -0.7092624  1.2111950
## 5  -1.0072522  1.05737478 0.3621886  0.5967832  0.5366266 -0.9254329 -0.2820031
## 7  -1.6400520 -2.21472131 0.3155253 -0.2315052 -0.4187554 -0.5304185 -0.6046161
## 10  1.9779028  0.41731956 1.4595861 -1.1188443 -0.2739387 -1.4386292 -1.2859206
##           PC8         PC9        PC10        PC11        PC12        PC13
## 1  -0.9052929  1.71609810 -0.08311642 -0.29184403  0.73084704 -0.47634060
## 2   1.9202414  0.69996476 -1.02984856  1.57495450 -0.74871067 -1.46646407
## 4  -0.1539142 -0.09841286  0.74362960 -0.04303572 -0.18538671  0.21015502
## 5   1.0447612  0.34658835  0.16335290  0.87871284  0.33237743 -1.34836804
## 7   0.1259354  1.47819959  1.03443538 -1.00579625 -0.07810924 -0.08641455
## 10  1.5794963  0.68971940  1.00936439  0.38137888 -0.15056542  0.16737648
##         PC14        PC15       PC16         PC17 LoanApproval
## 1  0.1087655  0.32735680  0.1689914 -0.110133990            0
## 2  2.0473178 -0.01342197 -0.4757043 -0.024372166            0
## 4  0.3515410  0.69375601 -0.1136902 -0.004692987            0
## 5  0.8278122 -0.28111519 -0.3390518 -0.008627405            0
## 7  0.5462269  0.25641592  0.1699593 -0.116860401            0
## 10 0.0282496  0.11095827 -0.1763522  0.003658458            0
```

The code trains a Decision Tree model (`dt_model`) on the `train_data` dataset, using several predictors and the `LoanApproval` target variable with the method `class` for classification. It then visualizes the trained decision tree using `rpart.plot`. The model is used to make predictions (`dt_predictions`) on the `test_data`, ensuring that both the predictions and actual values have the same factor levels. The confusion matrix (`dt_conf_matrix`) is calculated to evaluate the model's performance, and it is printed for review.

```r
#  Decision Tree Model
dt_model <- rpart(
  LoanApproval ~ Annual.Income + DTI + Credit.Score +
    Years.in.current.job + Credit.Utilization + Home.Ownership + Purpose,
  data = train_data,
  method = "class",
  control = rpart.control(cp = 0.01)
)

# Visualize
rpart.plot(dt_model, main = "Decision Tree for Loan Approval")
```




```r
# Test Data
dt_predictions <- predict(dt_model, test_data, type = "class")


test_data$LoanApproval <- factor(test_data$LoanApproval)
dt_predictions <- factor(dt_predictions, levels = levels(test_data$LoanApproval))

dt_conf_matrix <- confusionMatrix(dt_predictions, test_data$LoanApproval)

print(dt_conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 21434  6534
##          1   826  1360
##
##                Accuracy : 0.7559
##                  95% CI : (0.751, 0.7608)
##     No Information Rate : 0.7382
##     P-Value [Acc > NIR] : 9.587e-13
##
##                   Kappa : 0.1763
##
##  Mcnemar's Test P-Value : < 2.2e-16
```

```
##
##             Sensitivity : 0.9629
##             Specificity : 0.1723
##          Pos Pred Value : 0.7664
##          Neg Pred Value : 0.6221
##              Prevalence : 0.7382
##          Detection Rate : 0.7108
##    Detection Prevalence : 0.9275
##       Balanced Accuracy : 0.5676
##
##        'Positive' Class : 0
##
```

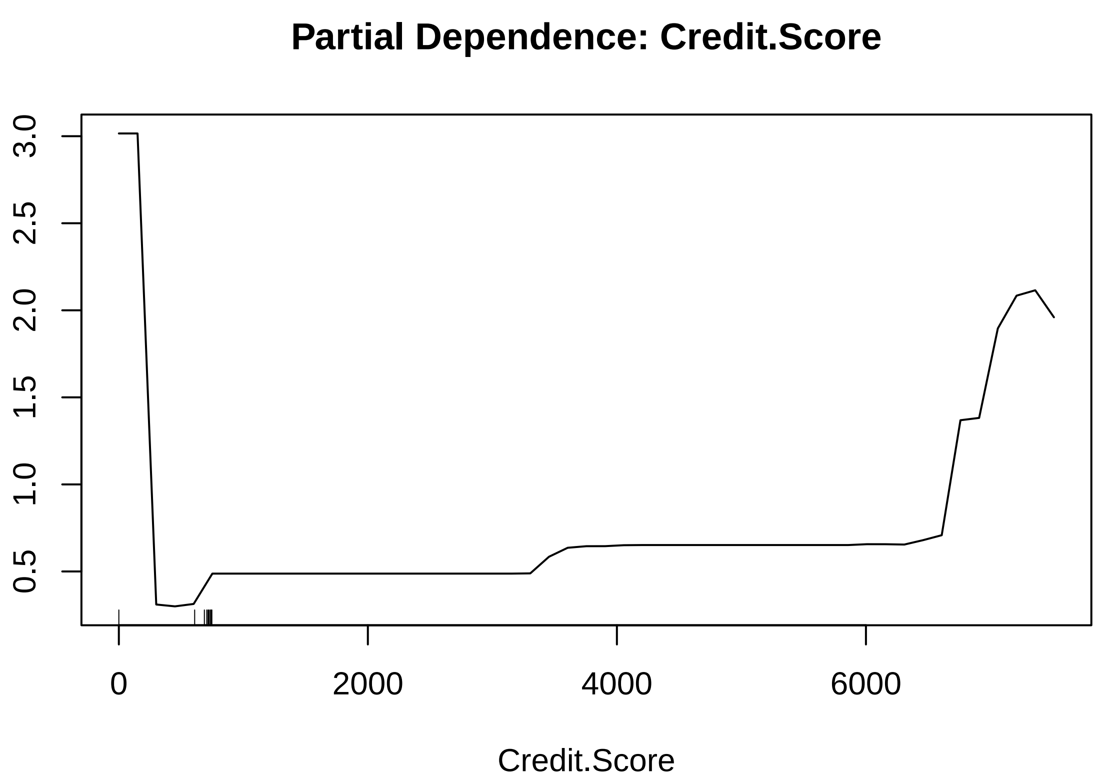
The code uses the train_data data set, a number of predictors, and the Loan Approval goal variable to train a Random Forest model (rf_model). The trained model (rf_predictions) is then used to forecast loan approval results on the test_data. The model's performance is assessed by converting the predictions to factors and computing a confusion matrix (rf_conf_matrix). Finally, a partial Credit dependency diagram.To illustrate how credit varies, a score feature is created.Score has an impact on the model's forecasts.

```r
# Random Forest Model
train_data$LoanApproval <- factor(train_data$LoanApproval)
test_data$LoanApproval <- factor(test_data$LoanApproval)

rf_model <- randomForest(
  LoanApproval ~ Annual.Income + DTI + Credit.Score + Years.in.current.job + Credit.Utilization + Home.
  data = train_data,
  ntree = 100
)

rf_predictions <- predict(rf_model, test_data)
rf_conf_matrix <- confusionMatrix(rf_predictions, test_data$LoanApproval)

# Test Data
rf_predictions <- predict(rf_model, test_data)

# Convert
rf_predictions <- factor(rf_predictions, levels = levels(test_data$LoanApproval))

# Confusion Matrix
rf_conf_matrix <- confusionMatrix(rf_predictions, test_data$LoanApproval)

print(rf_conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 20539  5300
##          1  1721  2594
##
##                Accuracy : 0.7672
##                  95% CI : (0.7623, 0.7719)
##     No Information Rate : 0.7382
##     P-Value [Acc > NIR] : < 2.2e-16
```

```
##
##                    Kappa : 0.2944
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##              Sensitivity : 0.9227
##              Specificity : 0.3286
##           Pos Pred Value : 0.7949
##           Neg Pred Value : 0.6012
##               Prevalence : 0.7382
##           Detection Rate : 0.6811
##     Detection Prevalence : 0.8569
##        Balanced Accuracy : 0.6256
##
##         'Positive' Class : 0
##
```

```
partialPlot(rf_model, train_data, Credit.Score, main = "Partial Dependence: Credit.Score")
```



Based on many predictors, the algorithm fits a Logistic Regression model (log_reg_model) to forecast LoanApproval. It models the likelihood of loan approval using the glm function with a binomial family. The test data is used to make the predictions, and anticipated probabilities greater than 0.5 are categorized as 1 (approved). The confusion matrix, or log_reg_conf_matrix, is used to evaluate the model's performance.

```
# Logistic Regression Model
log_reg_model <- glm(LoanApproval ~ Annual.Income + DTI +
```

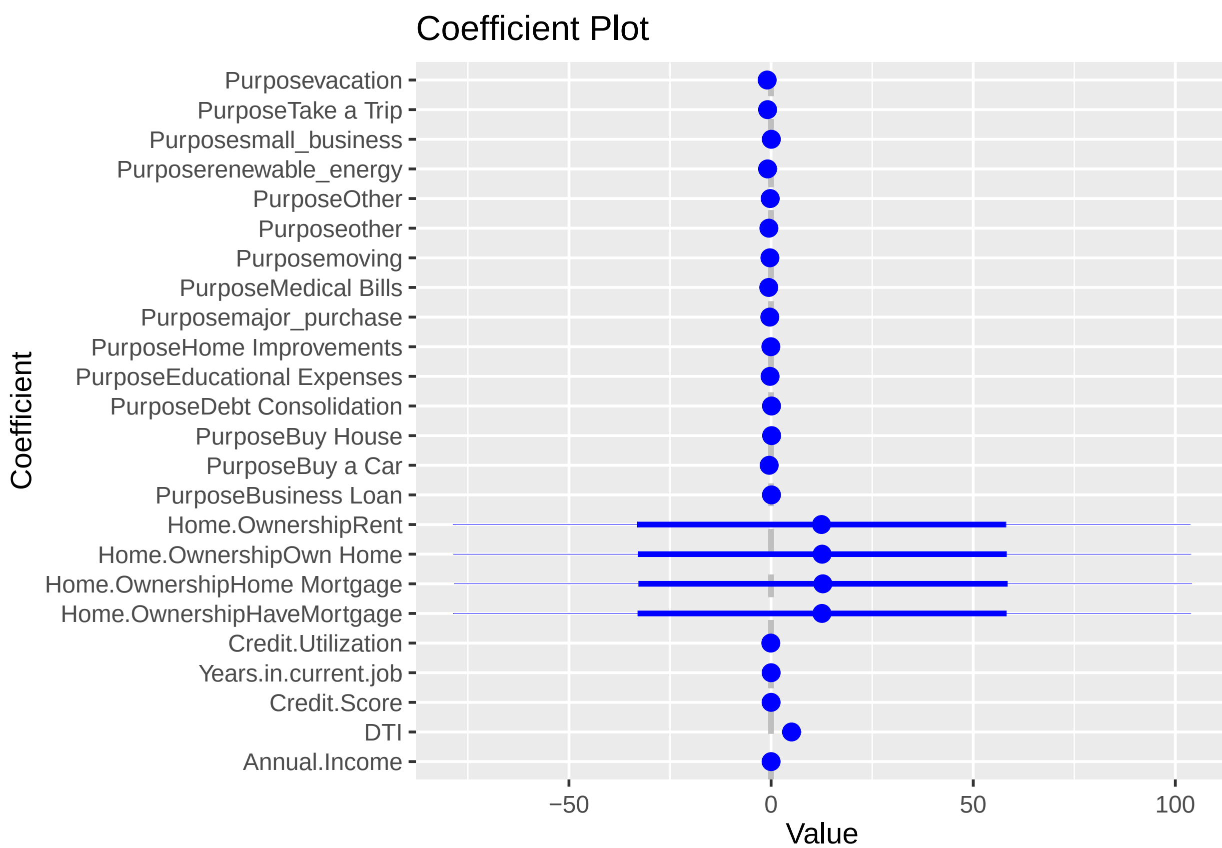
```r
Credit.Score + Years.in.current.job + Credit.Utilization +
Home.Ownership + Purpose,data = train_data, family = binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
log_reg_predictions <- predict(log_reg_model, test_data, type = "response")
log_reg_predictions <- ifelse(log_reg_predictions > 0.5, 1, 0)
log_reg_predictions <- factor(log_reg_predictions, levels = levels(test_data$LoanApproval))
log_reg_conf_matrix <- confusionMatrix(log_reg_predictions,
test_data$LoanApproval)

print(log_reg_conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 21600  6869
##          1   660  1025
##
##                Accuracy : 0.7503
##                  95% CI : (0.7454, 0.7552)
##     No Information Rate : 0.7382
##     P-Value [Acc > NIR] : 8.009e-07
##
##                   Kappa : 0.1343
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9704
##             Specificity : 0.1298
##          Pos Pred Value : 0.7587
##          Neg Pred Value : 0.6083
##              Prevalence : 0.7382
##          Detection Rate : 0.7163
##    Detection Prevalence : 0.9441
##       Balanced Accuracy : 0.5501
##
##        'Positive' Class : 0
##
```

```r
coefplot(log_reg_model, intercept = FALSE, main = "Feature Importance")
```




```r
# Extract Coefficients
coefficients <- summary(log_reg_model)$coefficients

# Data Frame
coeff_df <- data.frame(
  Predictor = rownames(coefficients),
  Estimate = coefficients[, "Estimate"]
)

# Remove the intercept
coeff_df <- coeff_df[coeff_df$Predictor != "(Intercept)", ]


# Bar Chart
ggplot(coeff_df, aes(x = reorder(Predictor, Estimate), y = Estimate)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +
  theme_minimal() +
  labs(title = "Logistic Regression Coefficients",
       x = "Predictor Variables",
       y = "Coefficient Estimate")
```

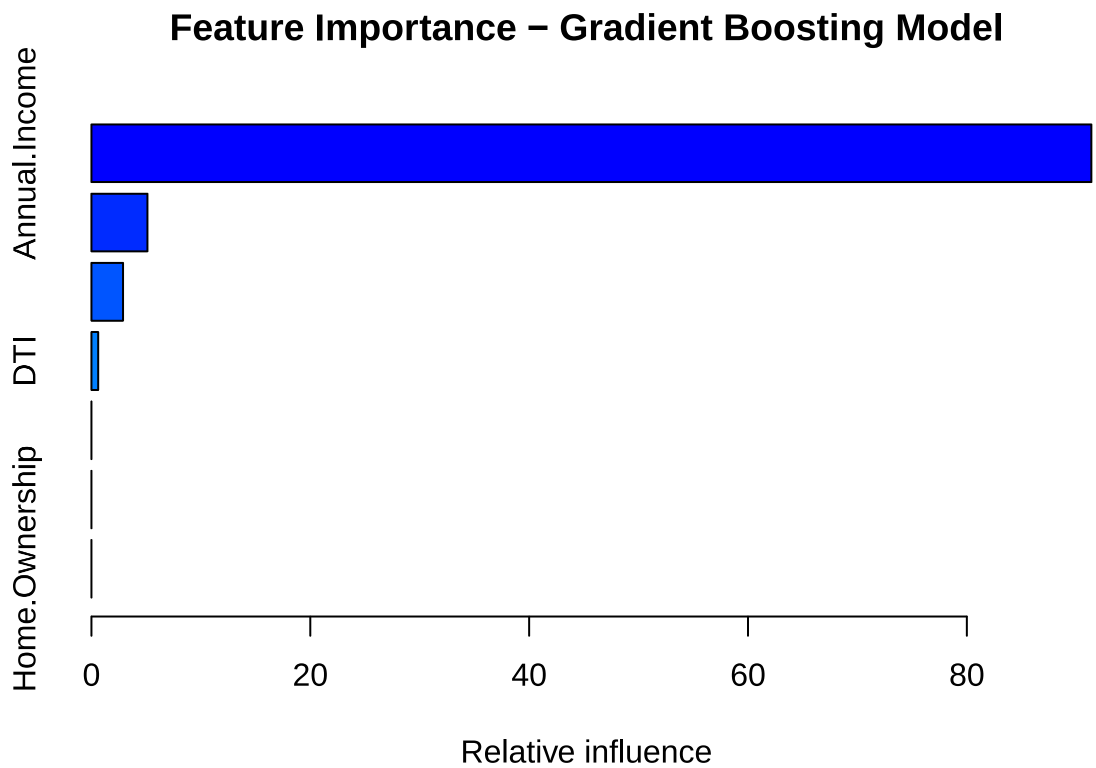
The goal variable, LoanApproval, is used to train the Gradient Boosting model (gb_model), which makes predictions about it using a number of features. The predict() method is used to make the model's predictions, and the results are classified using a threshold of 0.5. A confusion matrix is generated to evaluate the model's performance, comparing predicted values with actual values Lastly, the summary() function is used to illustrate the feature importance, illustrating how each predictor affects the model's predictions.

```
# Gradient Boosting Model
train_data$LoanApproval <- as.numeric(as.character(train_data$LoanApproval))
test_data$LoanApproval <- as.numeric(as.character(test_data$LoanApproval))

gb_model <- gbm(LoanApproval ~ Annual.Income + DTI +
Credit.Score + Years.in.current.job + Credit.Utilization +
  Home.Ownership + Purpose,
                data = train_data,
                distribution = "bernoulli",
                n.trees = 100,
                interaction.depth = 3,
                shrinkage = 0.01)

gb_predictions <- predict(gb_model, test_data,
n.trees = 100, type = "response")
gb_predictions <- ifelse(gb_predictions > 0.5, 1, 0)
gb_predictions <- factor(gb_predictions, levels = c(0, 1))
test_data$LoanApproval <- factor(test_data$LoanApproval,
levels = c(0, 1))
gb_conf_matrix <- confusionMatrix(gb_predictions,
test_data$LoanApproval)
```

```r
print(gb_conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 22260  7894
##          1     0     0
##
##                Accuracy : 0.7382
##                  95% CI : (0.7332, 0.7432)
##     No Information Rate : 0.7382
##     P-Value [Acc > NIR] : 0.503
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.7382
##          Neg Pred Value :    NaN
##              Prevalence : 0.7382
##          Detection Rate : 0.7382
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

```r
# Visualize
summary(gb_model, main = "Feature Importance - Gradient Boosting Model")
```




```
##                                        var    rel.inf
## Annual.Income                Annual.Income 91.3770695
## Credit.Score                  Credit.Score  5.1179462
## Purpose                            Purpose  2.8885583
## DTI                                    DTI  0.6164259
## Years.in.current.job Years.in.current.job  0.0000000
## Credit.Utilization      Credit.Utilization  0.0000000
## Home.Ownership              Home.Ownership  0.0000000
```

The Neural Network model (`ann_model`) is trained using the `nnet` package, with 5 hidden neurons and regularization (decay) to avoid overfitting. After training, predictions are made on the test data, and the results are converted to a factor for comparison with actual values. A confusion matrix (`ann_conf_matrix`) is generated to evaluate the model's performance. The `plotnet()` function is used to visualize the neural network structure, showcasing the relationships between input features and output predictions.

```
train_data$LoanApproval <- factor(train_data$LoanApproval, levels = c(0, 1))
test_data$LoanApproval <- factor(test_data$LoanApproval, levels = c(0, 1))

# Neural Network Model
set.seed(123)  # For reproducibility
ann_model <- nnet(
  LoanApproval ~ Annual.Income + DTI + Credit.Score + Years.in.current.job
  + Credit.Utilization + Home.Ownership + Purpose,
  data = train_data,
  size = 5,
  decay = 0.01,
  maxit = 200,
```

```r
  linout = FALSE
)
```

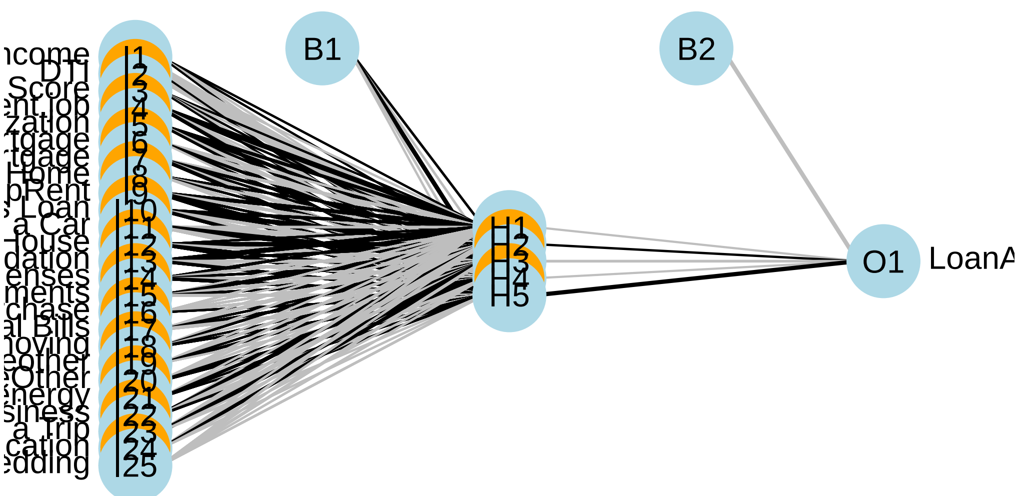
```
## # weights:  136
## initial  value 78482.868995
## iter  10 value 39751.944529
## iter  20 value 39668.526671
## iter  30 value 39662.235884
## iter  40 value 39656.701519
## iter  50 value 39648.746252
## iter  60 value 39644.556641
## iter  70 value 39642.774976
## iter  80 value 39048.095002
## iter  90 value 38011.545282
## iter 100 value 37114.957668
## iter 110 value 36819.761670
## iter 120 value 36760.455482
## iter 130 value 36707.053741
## iter 140 value 36643.812316
## final  value 36637.331325
## converged
```

```r
ann_predictions <- predict(ann_model, newdata = test_data, type = "class")

ann_predictions <- factor(ann_predictions,
levels = levels(test_data$LoanApproval))

# Confusion Matrix
ann_conf_matrix <- confusionMatrix(ann_predictions,
test_data$LoanApproval)
print(ann_conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 20420  5704
##          1  1840  2190
##
##                Accuracy : 0.7498
##                  95% CI : (0.7449, 0.7547)
##     No Information Rate : 0.7382
##     P-Value [Acc > NIR] : 2.116e-06
##
##                   Kappa : 0.2313
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9173
##             Specificity : 0.2774
##          Pos Pred Value : 0.7817
##          Neg Pred Value : 0.5434
##              Prevalence : 0.7382
##          Detection Rate : 0.6772
##    Detection Prevalence : 0.8664
```

```
##        Balanced Accuracy : 0.5974
##
##         'Positive' Class : 0
##
```

```r
# Visualize
plotnet(ann_model, circle_col = c("lightblue", "orange"))
```



Based on confusion matrices, the evaluate_model function computes evaluation metrics (Accuracy, Precision, Recall, and F1 Score) for various classification models. Confusion matrices of Random Forest, Logistic Regression, Gradient Boosting, Decision Tree, and Artificial Neural Network models are all subjected to the function. The final table including these measurements is printed to assess and compare the model performance after the metrics are saved in a data frame for comparison.

```r
# Function evaluation metrics
evaluate_model <- function(conf_matrix) {
  accuracy <- conf_matrix$overall['Accuracy']
  precision <- conf_matrix$byClass['Pos Pred Value']
  recall <- conf_matrix$byClass['Sensitivity']
  f1_score <- 2 * (precision * recall) / (precision + recall)

  return(list(
    Accuracy = accuracy,
    Precision = precision,
    Recall = recall,
    F1_Score = f1_score
  ))
```

```
}


rf_conf_matrix <- confusionMatrix(factor(rf_predictions), factor(test_data$LoanApproval))
log_reg_conf_matrix <- confusionMatrix(log_reg_predictions,
test_data$LoanApproval)
gb_conf_matrix <- confusionMatrix(gb_predictions, test_data$LoanApproval)
dt_conf_matrix <- confusionMatrix(factor(dt_predictions), factor(test_data$LoanApproval))
ann_conf_matrix <- confusionMatrix(ann_predictions, test_data$LoanApproval)

# Calculate
rf_metrics <- evaluate_model(rf_conf_matrix)
log_reg_metrics <- evaluate_model(log_reg_conf_matrix)
gb_metrics <- evaluate_model(gb_conf_matrix)
dt_metrics <- evaluate_model(dt_conf_matrix) # Added Decision Tree
ann_metrics <- evaluate_model(ann_conf_matrix)

# Combine
model_comparison <- data.frame(
  Model = c("Random Forest", "Logistic Regression",
"Gradient Boosting", "Decision Tree", "Artificial Neural Network"),
  Accuracy = c(rf_metrics$Accuracy, log_reg_metrics$Accuracy,
  gb_metrics$Accuracy, dt_metrics$Accuracy, ann_metrics$Accuracy),
  Precision = c(rf_metrics$Precision, log_reg_metrics$Precision, gb_metrics$Precision, dt_metrics$Precis
  Recall = c(rf_metrics$Recall, log_reg_metrics$Recall, gb_metrics$Recall, dt_metrics$Recall, ann_metric
  F1_Score = c(rf_metrics$F1_Score, log_reg_metrics$F1_Score,
gb_metrics$F1_Score, dt_metrics$F1_Score, ann_metrics$F1_Score))


print(model_comparison)
```

```
##                       Model  Accuracy Precision    Recall  F1_Score
## 1             Random Forest 0.7671619 0.7948837 0.9226864 0.8540302
## 2       Logistic Regression 0.7503150 0.7587200 0.9703504 0.8515839
## 3         Gradient Boosting 0.7382105 0.7382105 1.0000000 0.8493914
## 4             Decision Tree 0.7559196 0.7663759 0.9628931 0.8534682
## 5 Artificial Neural Network 0.7498176 0.7816567 0.9173405 0.8440807
```

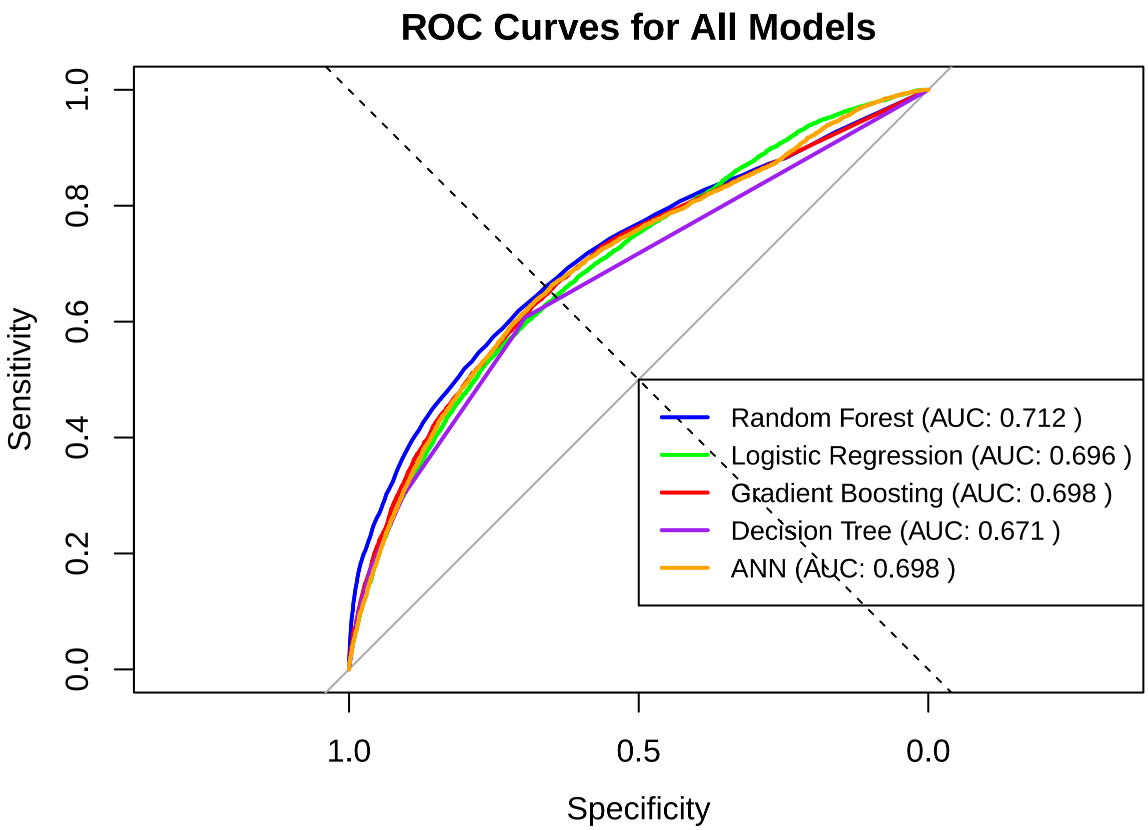
Using AUC (Area Under the Curve) from ROC (Receiver Operating Characteristic) analysis, the given code assesses many models (Random Forest, Logistic Regression, Gradient Boosting, Decision Tree, and Artificial Neural Network). The projected probability for each model's AUC values are computed and shown. The ROC curves for each model are plotted in various colors, with the baseline indicated by a diagonal line. The model names and associated AUC values are displayed in a legend that is appended to the plot. This makes it easier to compare the models' performances visually.

```
suppressMessages({
  suppressWarnings({
    rf_probabilities <- predict(rf_model, test_data, type = "prob")[, 2]
    rf_roc <- roc(test_data$LoanApproval, rf_probabilities)
    rf_auc <- auc(rf_roc)

    log_reg_probabilities <- predict(log_reg_model, test_data, type = "response")
    log_reg_roc <- roc(test_data$LoanApproval, log_reg_probabilities)
    log_reg_auc <- auc(log_reg_roc)
```

```r
    gb_probabilities <- predict(gb_model, test_data, n.trees = 100, type = "response")
    gb_roc <- roc(test_data$LoanApproval, gb_probabilities)
    gb_auc <- auc(gb_roc)

    dt_probabilities <- predict(dt_model, test_data, type = "prob")[, 2]
    dt_roc <- roc(test_data$LoanApproval, dt_probabilities)
    dt_auc <- auc(dt_roc)

    ann_probabilities <- predict(ann_model, test_data, type = "raw")
    ann_roc <- roc(test_data$LoanApproval, ann_probabilities)
    ann_auc <- auc(ann_roc)

    # Print the AUC for each model
    cat("Random Forest AUC: ", rf_auc, "\n")
    cat("Logistic Regression AUC: ", log_reg_auc, "\n")
    cat("Gradient Boosting AUC: ", gb_auc, "\n")
    cat("Decision Tree AUC: ", dt_auc, "\n")
    cat("Artificial Neural Network AUC: ", ann_auc, "\n")
  })
})
```

```
## Random Forest AUC:  0.7117709
## Logistic Regression AUC:  0.6964178
## Gradient Boosting AUC:  0.6975164
## Decision Tree AUC:  0.6713799
## Artificial Neural Network AUC:  0.6983226
```

```r
plot(rf_roc, col = "blue", main = "ROC Curves for All Models", lwd = 2)
lines(log_reg_roc, col = "green", lwd = 2)
lines(gb_roc, col = "red", lwd = 2)
lines(dt_roc, col = "purple", lwd = 2)
lines(ann_roc, col = "orange", lwd = 2)


abline(a = 0, b = 1, col = "black", lty = 2)

legend(x = 0.5, y = 0.5,
       legend = c(paste("Random Forest (AUC:", round(rf_auc, 3), ")"),
                  paste("Logistic Regression (AUC:", round(log_reg_auc, 3), ")"),
                  paste("Gradient Boosting (AUC:", round(gb_auc, 3), ")"),
                  paste("Decision Tree (AUC:", round(dt_auc, 3), ")"),
                  paste("ANN (AUC:", round(ann_auc, 3), ")")),
       col = c("blue", "green", "red", "purple", "orange"),
       lwd = 2,
       cex = 0.8)
```


ROC Curves for All Models
Sensitivity
Specificity
0.0
0.2
0.4
0.6
0.8
1.0
1.0
0.5
0.0
Random Forest (AUC: 0.712)
Logistic Regression (AUC: 0.696)
Gradient Boosting (AUC: 0.698)
Decision Tree (AUC: 0.671)
ANN (AUC: 0.698)