Customer Acquisition & Retention Analysis

AUTHOR
Elizabeth Obst, Shamir Cardenas, Jordan Bona, Olivia Shipley

PUBLISHED
November 22, 2025

▼ Code

```
library(SMCRM)
```

```
Warning: package 'SMCRM' was built under R version 4.5.2
```

▼ Code

```
library(randomForest)
```

```
Warning: package 'randomForest' was built under R version 4.5.2

randomForest 4.7-1.2

Type rfNews() to see new features/changes/bug fixes.
```

▼ Code

```
library(rpart)
```

```
Warning: package 'rpart' was built under R version 4.5.2
```

▼ Code

```
library(rpart.plot)
```

```
Warning: package 'rpart.plot' was built under R version 4.5.2
```

▼ Code

```
library(pdp)
```

```
Warning: package 'pdp' was built under R version 4.5.2
```

▼ Code

```
library(pROC)
```

```
Warning: package 'pROC' was built under R version 4.5.1

Type 'citation("pROC")' for a citation.
```

```
Attaching package: 'pROC'

The following objects are masked from 'package:stats':

    cov, smooth, var
```

▼ Code

```
library(corrplot)
```

```
Warning: package 'corrplot' was built under R version 4.5.1

corrplot 0.95 loaded
```

▼ Code

```
# Set seed so results are reproducible
set.seed(123)
```

▼ Code

```
data(acquisitionRetention)

cat("Dataset dimensions:", nrow(acquisitionRetention), "rows,",
    ncol(acquisitionRetention), "columns\n\n")
```

```
Dataset dimensions: 500 rows, 15 columns
```

▼ Code

```
str(acquisitionRetention)
```

```
'data.frame':   500 obs. of  15 variables:
 $ customer   : num  1 2 3 4 5 6 7 8 9 10 ...
 $ acquisition: num  1 1 1 0 1 1 1 1 0 0 ...
 $ duration   : num  1635 1039 1288 0 1631 ...
 $ profit     : num  6134 3524 4081 -638 5446 ...
 $ acq_exp    : num  694 460 249 638 589 ...
 $ ret_exp    : num  972 450 805 0 920 ...
 $ acq_exp_sq : num  480998 211628 62016 407644 346897 ...
 $ ret_exp_sq : num  943929 202077 648089 0 846106 ...
 $ freq       : num  6 11 21 0 2 7 15 13 0 0 ...
 $ freq_sq    : num  36 121 441 0 4 49 225 169 0 0 ...
 $ crossbuy   : num  5 6 6 0 9 4 5 5 0 0 ...
 $ sow        : num  95 22 90 0 80 48 51 23 0 0 ...
 $ industry   : num  1 0 0 0 0 1 0 1 0 1 ...
 $ revenue    : num  47.2 45.1 29.1 40.6 48.7 ...
 $ employees  : num  898 686 1423 181 631 ...
```

EDA

Basic Statistics

▼ Code

```
# Summary statistics
summary(acquisitionRetention)
```

```
   customer      acquisition       duration          profit       
Min.   :  1.0   Min.   :0.000   Min.   :   0.0   Min.   :-1027.0  
1st Qu.:125.8   1st Qu.:0.000   1st Qu.:   0.0   1st Qu.: -316.3  
Median :250.5   Median :1.000   Median : 957.5   Median : 3369.9  
Mean   :250.5   Mean   :0.676   Mean   : 742.5   Mean   : 2403.8  
3rd Qu.:375.2   3rd Qu.:1.000   3rd Qu.:1146.2   3rd Qu.: 3931.6  
Max.   :500.0   Max.   :1.000   Max.   :1673.0   Max.   : 6134.3  
   acq_exp           ret_exp         acq_exp_sq          ret_exp_sq     
Min.   :   1.21   Min.   :   0.0   Min.   :1.460e+00   Min.   :      0  
1st Qu.: 384.14   1st Qu.:   0.0   1st Qu.:1.476e+05   1st Qu.:      0  
Median : 491.66   Median : 398.1   Median :2.417e+05   Median : 158480  
Mean   : 493.35   Mean   : 336.3   Mean   :2.712e+05   Mean   : 184000  
3rd Qu.: 600.21   3rd Qu.: 514.3   3rd Qu.:3.602e+05   3rd Qu.: 264466  
Max.   :1027.04   Max.   :1095.0   Max.   :1.055e+06   Max.   :1198937  
     freq          freq_sq          crossbuy           sow        
Min.   : 0.00   Min.   :  0.00   Min.   : 0.000   Min.   :  0.00  
1st Qu.: 0.00   1st Qu.:  0.00   1st Qu.: 0.000   1st Qu.:  0.00  
Median : 6.00   Median : 36.00   Median : 5.000   Median : 44.00  
Mean   : 6.22   Mean   : 69.25   Mean   : 4.052   Mean   : 38.88  
3rd Qu.:11.00   3rd Qu.:121.00   3rd Qu.: 7.000   3rd Qu.: 66.00  
Max.   :21.00   Max.   :441.00   Max.   :11.000   Max.   :116.00  
   industry        revenue        employees     
Min.   :0.000   Min.   :14.49   Min.   :  18.0  
1st Qu.:0.000   1st Qu.:33.53   1st Qu.: 503.0  
Median :1.000   Median :41.43   Median : 657.5  
Mean   :0.522   Mean   :40.54   Mean   : 671.5  
3rd Qu.:1.000   3rd Qu.:47.52   3rd Qu.: 826.0  
Max.   :1.000   Max.   :65.10   Max.   :1461.0  
```

▼ Code

```
# Check for missing values
cat("\nMissing values per column:\n")
```

```

Missing values per column:
```

▼ Code

```
missing_counts <- colSums(is.na(acquisitionRetention))
print(missing_counts[missing_counts > 0])
```

```
named numeric(0)
```

▼ Code

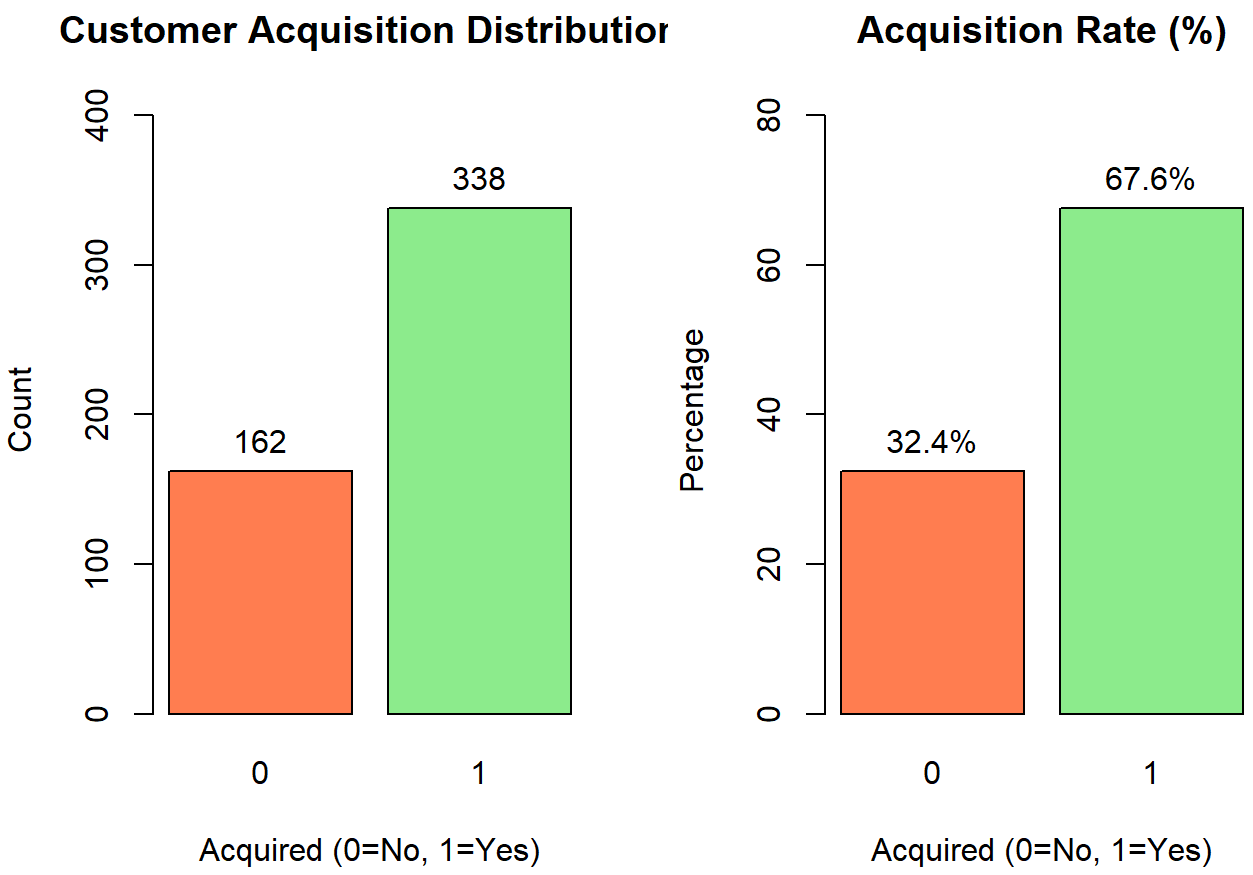
```
if(sum(missing_counts) == 0) {
  cat("No missing values found!\n")
}
```

```
No missing values found!
```

Target Variable Analysis

▼ Code

```
par(mfrow = c(1, 2))

# Acquisition counts
acq_table <- table(acquisitionRetention$acquisition)
barplot(acq_table,
        main = "Customer Acquisition Distribution",
        xlab = "Acquired (0=No, 1=Yes)",
        ylab = "Count",
        col = c("coral", "lightgreen"),
        ylim = c(0, max(acq_table) * 1.2))
text(x = c(0.7, 1.9), y = acq_table, labels = acq_table, pos = 3)

# Acquisition percentages
acq_prop <- prop.table(acq_table) * 100
barplot(acq_prop,
        main = "Acquisition Rate (%)",
        xlab = "Acquired (0=No, 1=Yes)",
        ylab = "Percentage",
        col = c("coral", "lightgreen"),
        ylim = c(0, max(acq_prop) * 1.2))
text(x = c(0.7, 1.9), y = acq_prop,
     labels = paste0(round(acq_prop, 1), "%"), pos = 3)
```



▼ Code

```
par(mfrow = c(1, 1))

cat("\nAcquisition Summary:\n")
```

```
Acquisition Summary:
```

▼ Code

```
cat("Not Acquired:", acq_table[1], "(", round(acq_prop[1], 1), "%)\n")
```

```
Not Acquired: 162 ( 32.4 %)
```

▼ Code

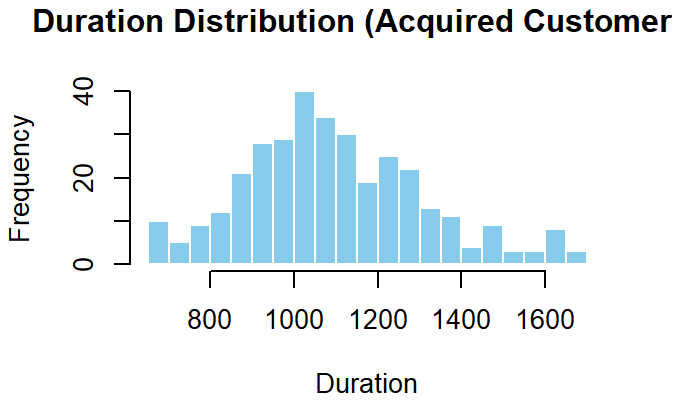
```
cat("Acquired:", acq_table[2], "(", round(acq_prop[2], 1), "%)\n")
```

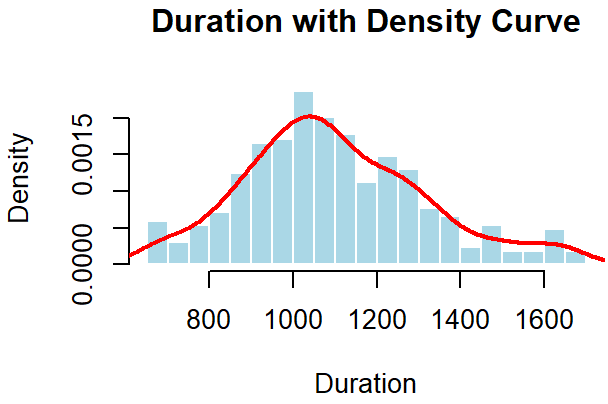
```
Acquired: 338 ( 67.6 %)
```

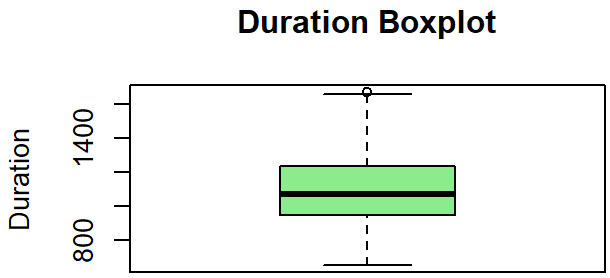
Duration Analysis

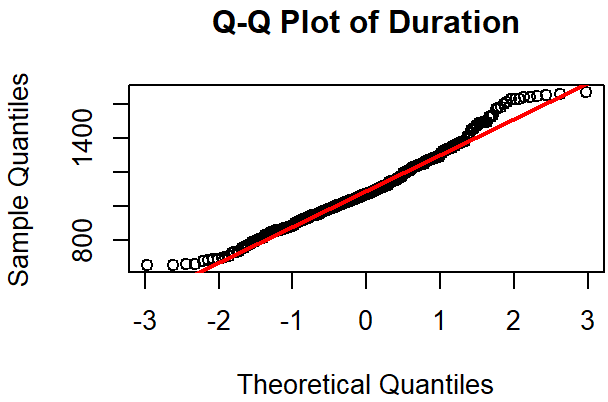
▼ Code

```
# Analyze duration for acquired customers
acquired <- acquisitionRetention[acquisitionRetention$acquisition == 1, ]
not_acquired <- acquisitionRetention[acquisitionRetention$acquisition == 0, ]

par(mfrow = c(2, 2))

# Duration for acquired customers
hist(acquired$duration,
     main = "Duration Distribution (Acquired Customers)",
     xlab = "Duration",
     col = "skyblue",
     breaks = 30,
     border = "white")

# Duration with density curve
hist(acquired$duration,
     probability = TRUE,
     main = "Duration with Density Curve",
     xlab = "Duration",
     col = "lightblue",
     breaks = 30,
     border = "white")
lines(density(acquired$duration), col = "red", lwd = 2)

# Boxplot
boxplot(acquired$duration,
        main = "Duration Boxplot",
        ylab = "Duration",
        col = "lightgreen",
        horizontal = FALSE)

# QQ plot to check normality
qqnorm(acquired$duration, main = "Q-Q Plot of Duration")
qqline(acquired$duration, col = "red", lwd = 2)
```









▼ Code

```
par(mfrow = c(1, 1))

cat("\nDuration Summary Statistics (Acquired Customers):\n")
```

```
Duration Summary Statistics (Acquired Customers):
```

▼ Code

```
print(summary(acquired$duration))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  654.0   950.5  1072.0  1098.3  1235.8  1673.0 
```

▼ Code

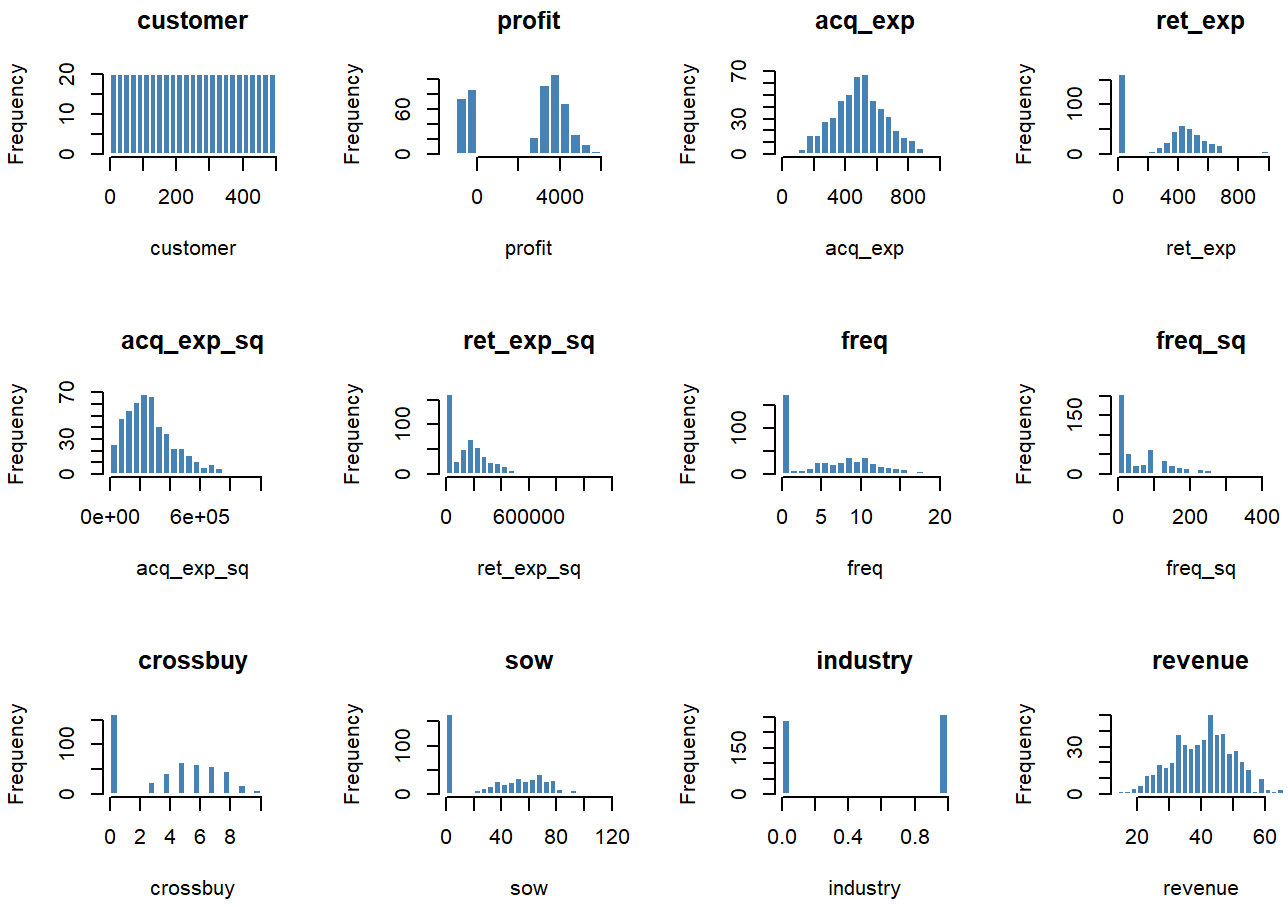
```
cat("\nStandard Deviation:", sd(acquired$duration), "\n")
```

```
Standard Deviation: 216.713
```

Feature Distributions

▼ Code

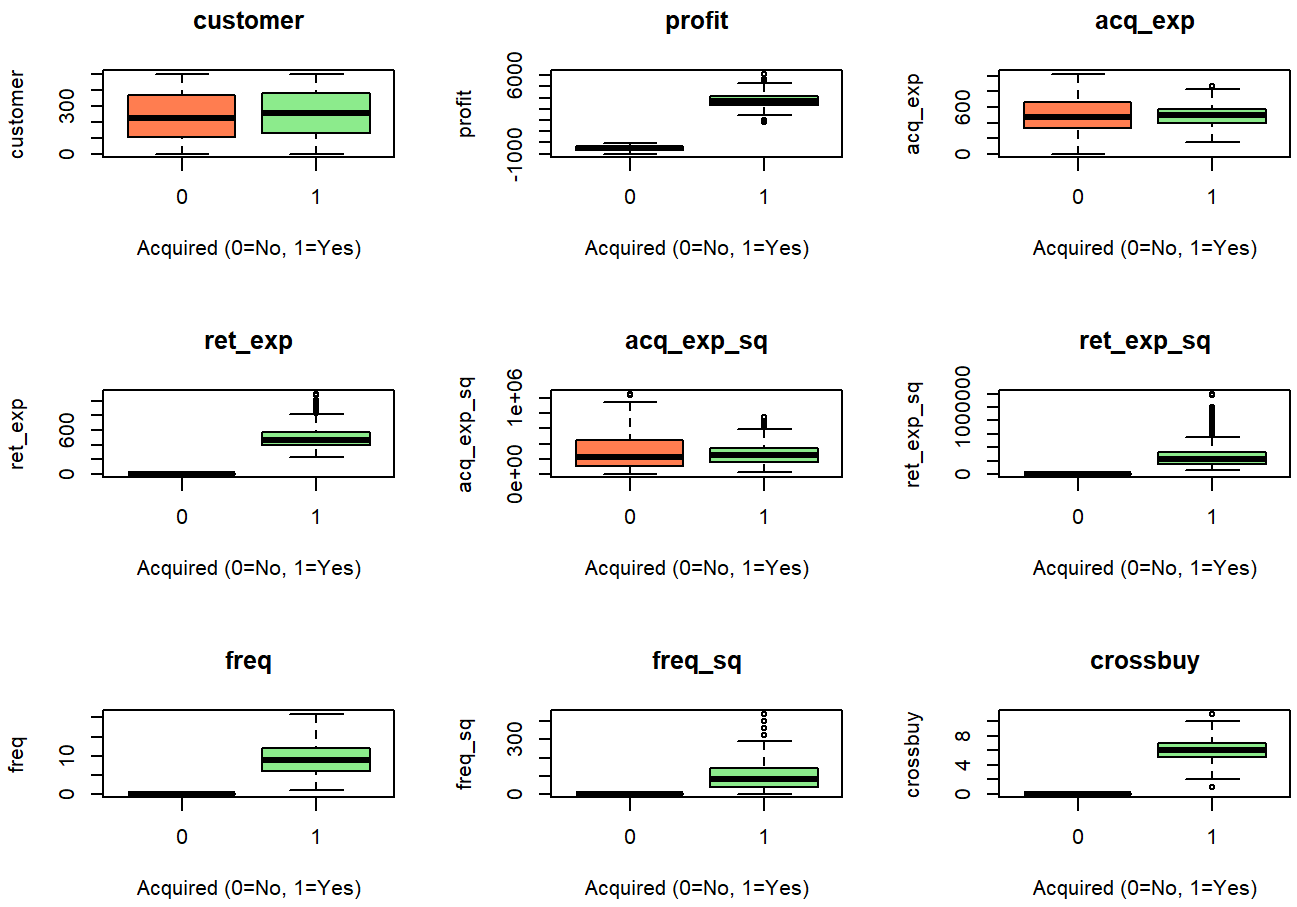
```
# Get all numeric predictors
numeric_cols <- names(acquisitionRetention)[sapply(acquisitionRetention, is.numeric)]
numeric_cols <- setdiff(numeric_cols, c("acquisition", "duration"))

if(length(numeric_cols) > 0) {
  n_plots <- min(12, length(numeric_cols))
  par(mfrow = c(3, 4))

  for(i in 1:n_plots) {
    col_name <- numeric_cols[i]
    hist(acquisitionRetention[[col_name]],
         main = col_name,
         xlab = col_name,
         col = "steelblue",
         breaks = 20,
         border = "white")
  }

  par(mfrow = c(1, 1))
}
```



Features by Acquisition Status

▼ Code

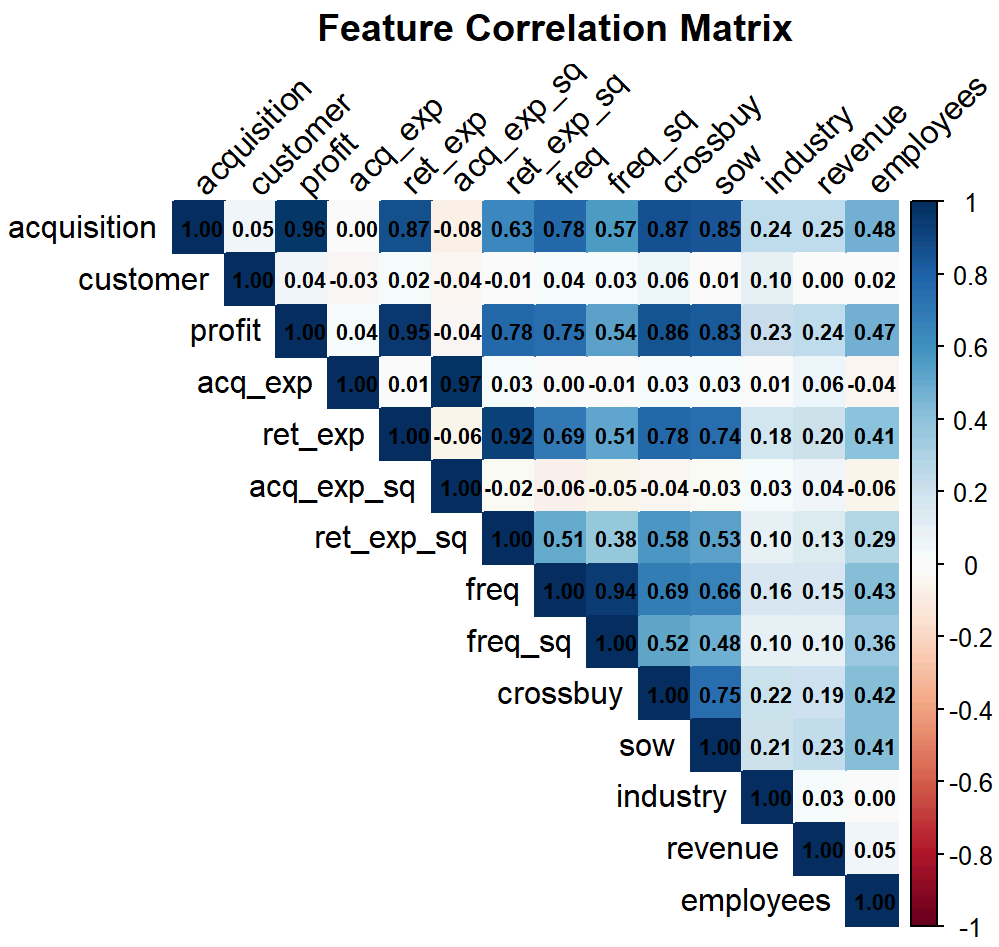
```
# Compare feature distributions between acquired and not acquired
if(length(numeric_cols) > 0) {
  n_plots <- min(9, length(numeric_cols))
  par(mfrow = c(3, 3))

  for(i in 1:n_plots) {
    col_name <- numeric_cols[i]
    boxplot(acquisitionRetention[[col_name]] ~ acquisitionRetention$acquisition,
            main = col_name,
            xlab = "Acquired (0=No, 1=Yes)",
            ylab = col_name,
            col = c("coral", "lightgreen"),
            outline = TRUE)
  }

  par(mfrow = c(1, 1))
}
```



Correlation Analysis

▼ Code

```
# Create correlation matrix
numeric_data <- acquisitionRetention[, c("acquisition", numeric_cols[1:min(15, length(numeric_cols
cor_matrix <- cor(numeric_data, use = "complete.obs")

# Plot correlation heatmap
corrplot(cor_matrix,
         method = "color",
         type = "upper",
         tl.col = "black",
         tl.srt = 45,
         addCoef.col = "black",
         number.cex = 0.7,
         title = "Feature Correlation Matrix",
         mar = c(0, 0, 2, 0))
```



▼ Code

```
# Find highly correlated features with acquisition
cat("\nCorrelation with Acquisition:\n")
```

```
Correlation with Acquisition:
```

▼ Code

```
acq_cor <- sort(abs(cor_matrix[, "acquisition"]), decreasing = TRUE)
print(head(acq_cor, 10))
```

```
acquisition      profit     ret_exp    crossbuy         sow        freq 
  1.0000000   0.9639299   0.8741157   0.8661546   0.8471508   0.7789100 
 ret_exp_sq     freq_sq   employees     revenue 
  0.6313023   0.5676057   0.4770195   0.2488373 
```

Data Preparation

▼ Code

```
# Check the data structure
cat("All columns:\n")
```

```
All columns:
```

▼ Code

```
print(names(acquisitionRetention))
```

```
 [1] "customer"    "acquisition" "duration"    "profit"      "acq_exp"
 [6] "ret_exp"     "acq_exp_sq"  "ret_exp_sq"  "freq"        "freq_sq"
[11] "crossbuy"    "sow"         "industry"    "revenue"     "employees"
```

▼ Code

```
# Variables to remove (consequences of acquisition, not predictors):
leakage_vars <- c("duration", "profit", "ret_exp", "acq_exp_sq", "ret_exp_sq",
                  "freq", "freq_sq", "crossbuy", "sow")

cat("Removing these variables to prevent leakage:\n")
```

```
Removing these variables to prevent leakage:
```

▼ Code

```
print(leakage_vars)
```

```
[1] "duration"   "profit"     "ret_exp"    "acq_exp_sq" "ret_exp_sq"
[6] "freq"       "freq_sq"    "crossbuy"   "sow"
```

▼ Code

```
# Check for ID columns
id_cols <- grep("id|ID|customer", names(acquisitionRetention),
                ignore.case = TRUE, value = TRUE)
if(length(id_cols) > 0) {
  cat("\nAlso removing ID columns:", paste(id_cols, collapse = ", "), "\n")
  leakage_vars <- c(leakage_vars, id_cols)
}
```

```

Also removing ID columns: customer
```

▼ Code

```
# Create modeling dataset with only legitimate predictors
data_for_models <- acquisitionRetention[, !names(acquisitionRetention) %in% leakage_vars]
```

```
cat("\nVariables Used for Modeling:\n")
```

```
Variables Used for Modeling:
```

▼ Code

```
cat("Features:", paste(setdiff(names(data_for_models), "acquisition"), collapse = ", "), "\n")
```

```
Features: acq_exp, industry, revenue, employees
```

▼ Code

```
cat("Number of features:", ncol(data_for_models) - 1, "\n")
```

```
Number of features: 4
```

▼ Code

```
# Split data 70/30
set.seed(123)
n <- nrow(data_for_models)
train_size <- floor(0.7 * n)
train_indices <- sample(1:n, train_size)

train <- data_for_models[train_indices, ]
test <- data_for_models[-train_indices, ]

cat("\nTraining set:", nrow(train), "observations\n")
```

```
Training set: 350 observations
```

▼ Code

```
cat("Test set:", nrow(test), "observations\n")
```

```
Test set: 150 observations
```

▼ Code

```
cat("Acquisition rate in training:", round(mean(train$acquisition), 3), "\n")
```

```
Acquisition rate in training: 0.7
```

▼ Code

```
cat("Acquisition rate in test:", round(mean(test$acquisition), 3), "\n")
```

```
Acquisition rate in test: 0.62
```

Modeling

Random Forest Model

▼ Code

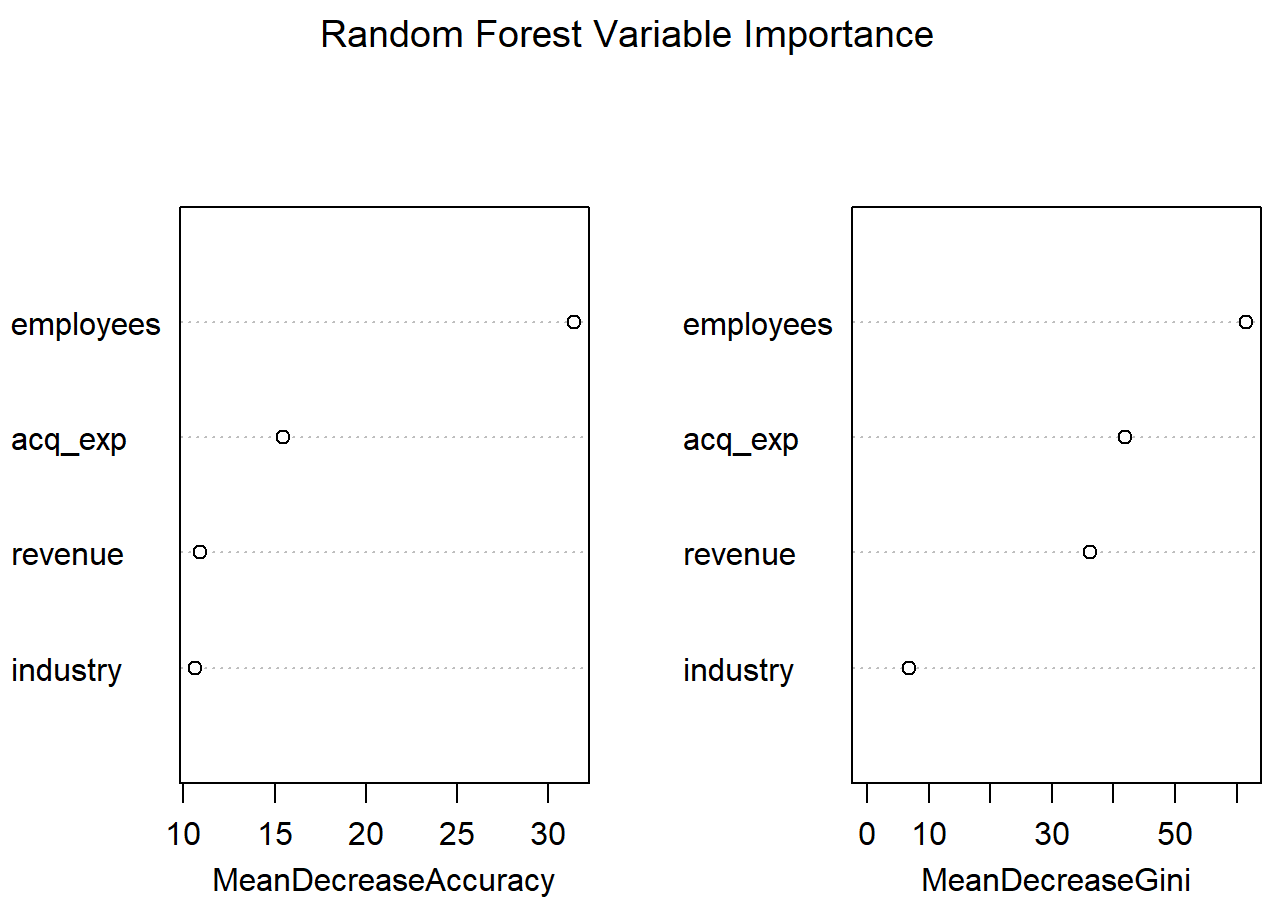
```
# Prepare data (remove duration column)
train_no_duration <- train
test_no_duration <- test

# Build Random Forest
rf_model <- randomForest(
  as.factor(acquisition) ~ .,
  data = train_no_duration,
  ntree = 300,
  importance = TRUE
)

print(rf_model)
```

```
Call:
 randomForest(formula = as.factor(acquisition) ~ ., data = train_no_duration,      ntree = 300,
importance = TRUE)
               Type of random forest: classification
                     Number of trees: 300
No. of variables tried at each split: 2

        OOB estimate of  error rate: 21.43%
Confusion matrix:
   0   1 class.error
0 61  44   0.4190476
1 31 214   0.1265306
```

Variable Importance

▼ Code

```
# Plot variable importance
varImpPlot(rf_model,
           main = "Random Forest Variable Importance",
           n.var = min(20, nrow(importance(rf_model))))
```



▼ Code

```
# Get importance scores
importance_scores <- importance(rf_model)
importance_df <- data.frame(
  Variable = rownames(importance_scores),
  Importance = importance_scores[, "MeanDecreaseGini"]
)
importance_df <- importance_df[order(-importance_df$Importance), ]

cat("\nTop 10 Most Important Variables:\n")
```

```
Top 10 Most Important Variables:
```

▼ Code

```
print(head(importance_df, 10))
```

```
           Variable Importance
employees employees  61.512949
acq_exp     acq_exp  41.862147
```

```
revenue     revenue  36.208863
industry   industry   6.749492
```

Hyperparameter Tuning

▼ Code

```
# Try different mtry values
cat("Tuning mtry parameter\n\n")
```

```
Tuning mtry parameter
```

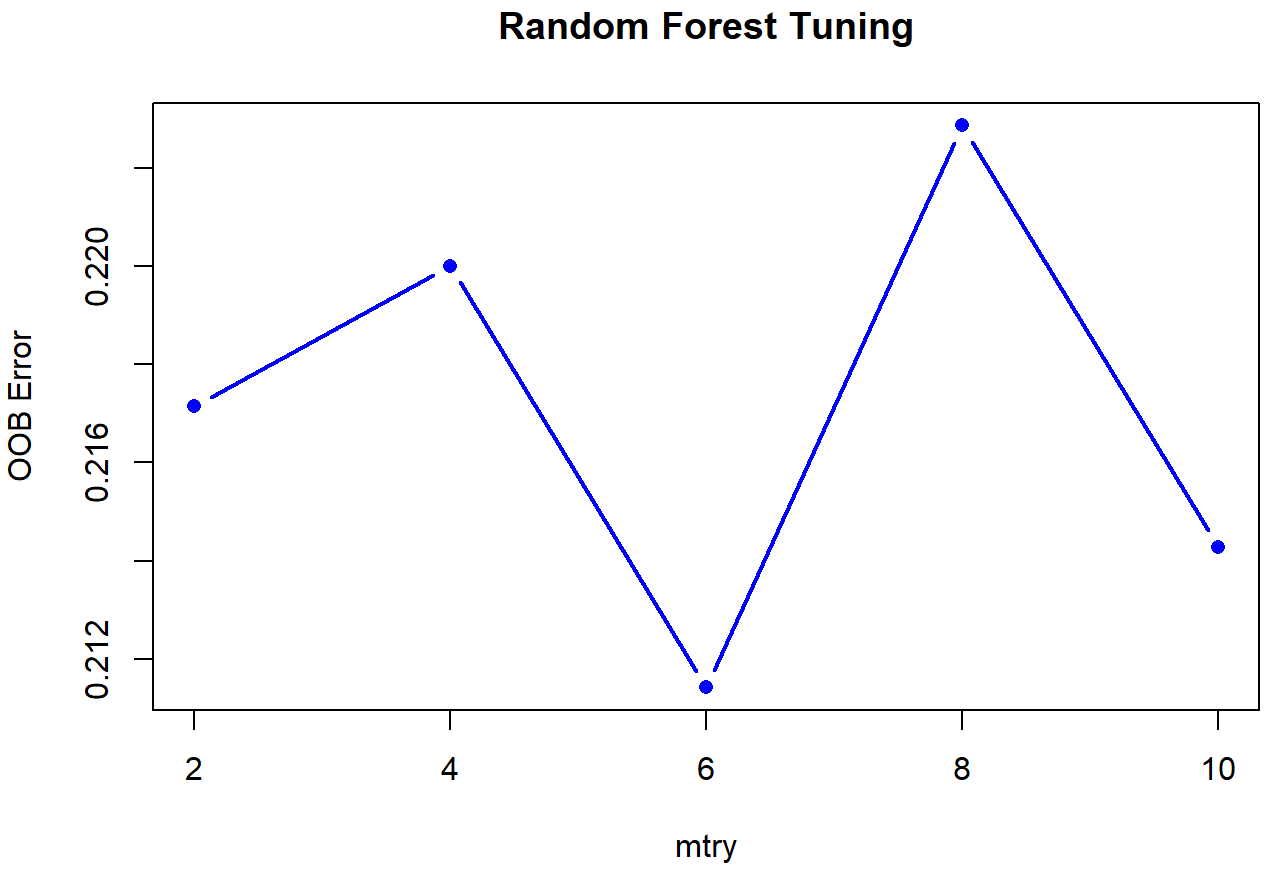
▼ Code

```
results <- data.frame(mtry = numeric(), OOB_error = numeric())

for(m in c(2, 4, 6, 8, 10)) {
  rf_temp <- randomForest(
    as.factor(acquisition) ~ .,
    data = train_no_duration,
    ntree = 300,
    mtry = m
  )

  oob_error <- rf_temp$err.rate[300, 1]
  results <- rbind(results, data.frame(mtry = m, OOB_error = oob_error))
  cat("mtry =", m, "| OOB Error =", round(oob_error, 4), "\n")
}
```

```
mtry = 2 | OOB Error = 0.2171
mtry = 4 | OOB Error = 0.22

Warning in randomForest.default(m, y, ...): invalid mtry: reset to within valid
range

mtry = 6 | OOB Error = 0.2114

Warning in randomForest.default(m, y, ...): invalid mtry: reset to within valid
range

mtry = 8 | OOB Error = 0.2229

Warning in randomForest.default(m, y, ...): invalid mtry: reset to within valid
range

mtry = 10 | OOB Error = 0.2143
```

▼ Code

```
# Plot tuning results
plot(results$mtry, results$OOB_error,
     type = "b",
     xlab = "mtry",
     ylab = "OOB Error",
     main = "Random Forest Tuning",
     col = "blue",
     pch = 16,
     lwd = 2)
```



▼ Code

```
best_mtry <- results$mtry[which.min(results$OOB_error)]
cat("\nBest mtry:", best_mtry, "\n")
```

```
Best mtry: 6
```

Final Random Forest Model

▼ Code

```
# Build final model with best parameters
rf_final <- randomForest(
  as.factor(acquisition) ~ .,
  data = train_no_duration,
  ntree = 300,
  mtry = best_mtry,
  importance = TRUE
)
```

```
Warning in randomForest.default(m, y, ...): invalid mtry: reset to within valid
range
```

▼ Code

```
cat("Final Random Forest Model:\n")
```

```
Final Random Forest Model:
```

▼ Code

```
print(rf_final)
```

```
Call:
 randomForest(formula = as.factor(acquisition) ~ ., data = train_no_duration,      ntree = 300,
mtry = best_mtry, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 300
No. of variables tried at each split: 4

        OOB estimate of  error rate: 22.86%
Confusion matrix:
   0   1 class.error
0 60  45   0.4285714
1 35 210   0.1428571
```

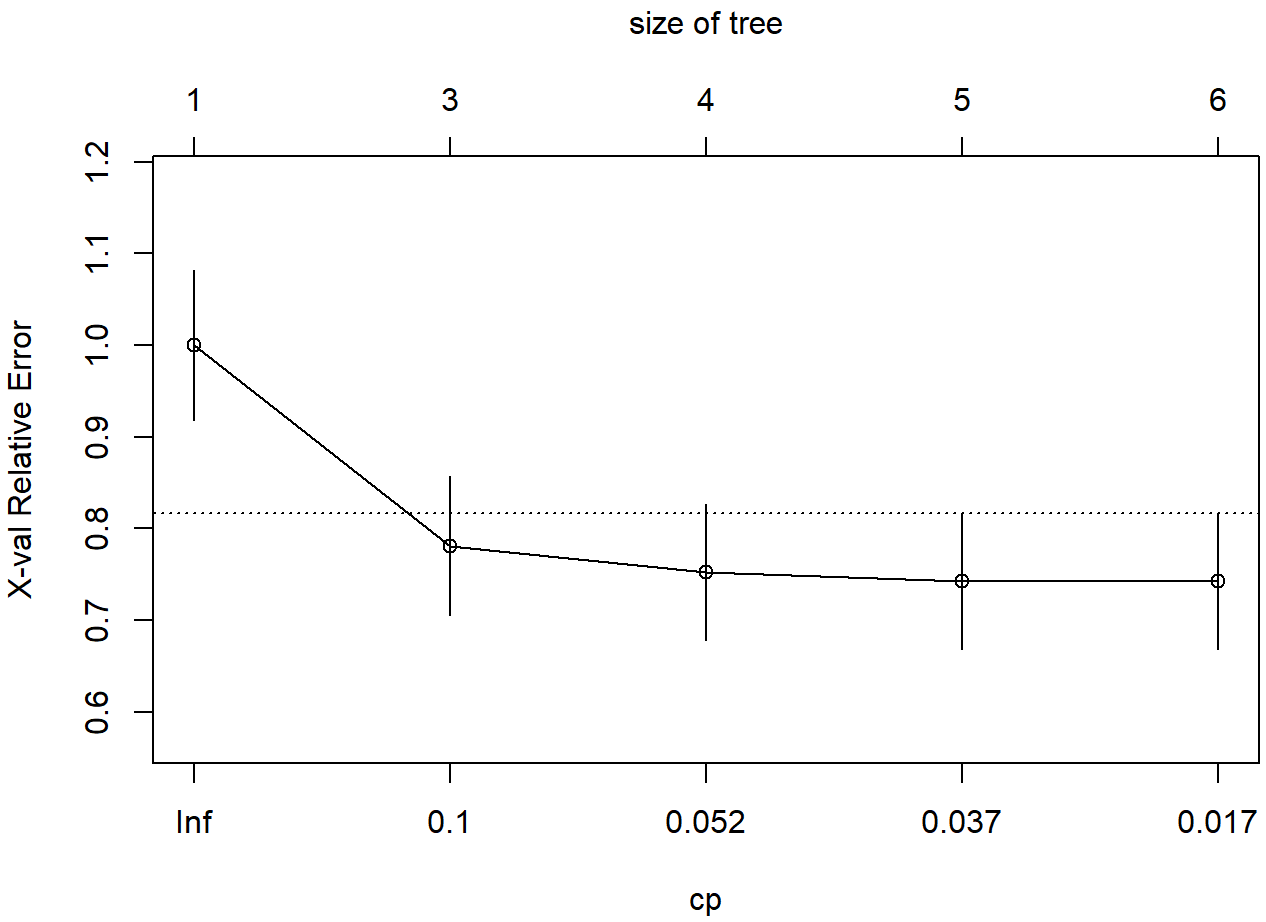
▼ Code

```
# Make predictions
rf_pred_class <- predict(rf_final, test_no_duration)
rf_pred_prob <- predict(rf_final, test_no_duration, type = "prob")[, 2]
rf_pred_numeric <- as.numeric(as.character(rf_pred_class))
```

Decision Tree Model

▼ Code

```
# Build decision tree
tree_model <- rpart(
  acquisition ~ .,
  data = train_no_duration,
  method = "class"
)

# Plot complexity parameter
plotcp(tree_model)
```



▼ Code

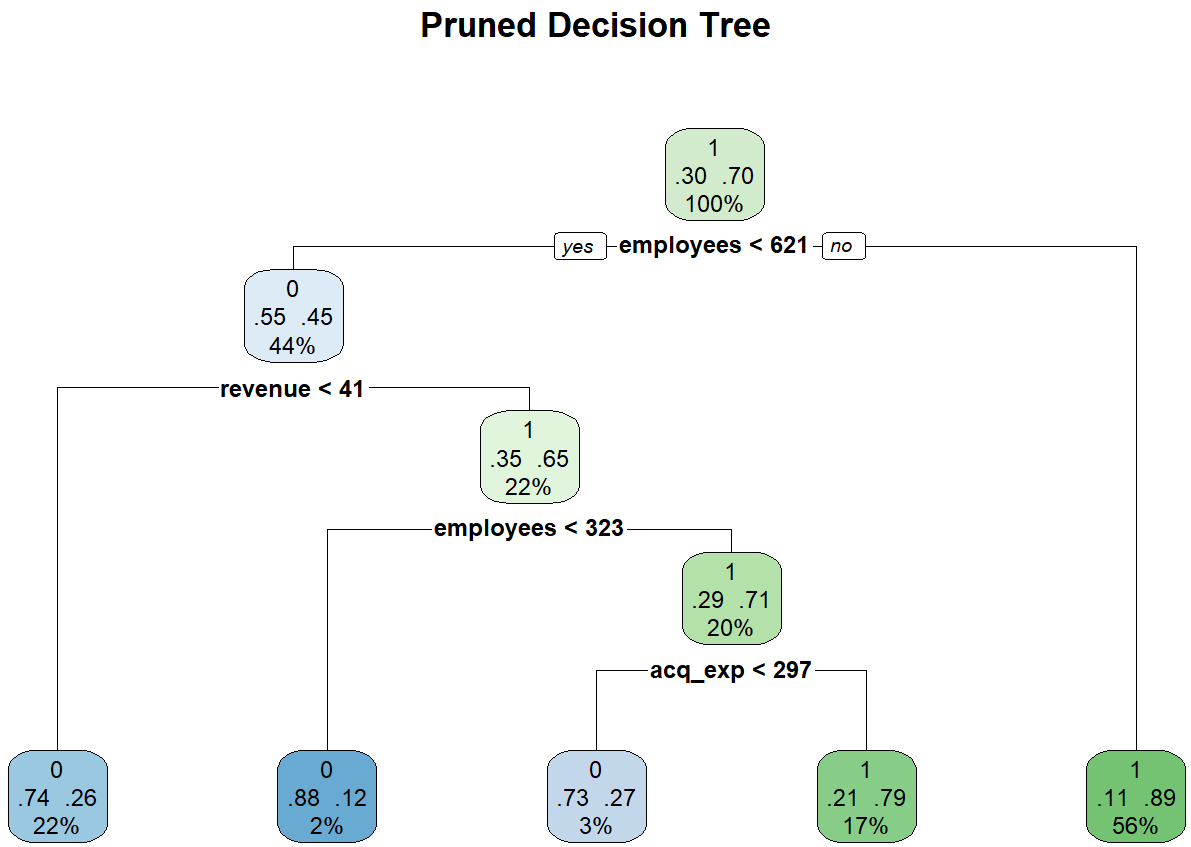
```
# Prune tree
best_cp <- tree_model$cptable[which.min(tree_model$cptable[, "xerror"]), "CP"]
cat("Best CP:", best_cp, "\n")
```

```
Best CP: 0.02857143
```

▼ Code

```
tree_pruned <- prune(tree_model, cp = best_cp)

# Visualize tree
```

```
rpart.plot(tree_pruned,
           main = "Pruned Decision Tree",
           extra = 104,
           fallen.leaves = TRUE)
```



▼ Code

```
# Make predictions
tree_pred_prob <- predict(tree_pruned, test_no_duration)[, 2]
tree_pred_class <- ifelse(tree_pred_prob > 0.5, 1, 0)
```

Logistic Regression Model

▼ Code

```
# Build logistic regression
logit_model <- glm(
  acquisition ~ .,
  data = train_no_duration,
  family = binomial
)
```

```
summary(logit_model)
```

```
Call:
glm(formula = acquisition ~ ., family = binomial, data = train_no_duration)

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -7.6871800  1.0653048  -7.216 5.36e-13 ***
acq_exp      0.0008318  0.0008680   0.958    0.338
industry     1.2727217  0.3073514   4.141 3.46e-05 ***
revenue      0.0869960  0.0161776   5.378 7.55e-08 ***
employees    0.0066901  0.0008758   7.639 2.20e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 427.61  on 349  degrees of freedom
Residual deviance: 285.54  on 345  degrees of freedom
AIC: 295.54

Number of Fisher Scoring iterations: 5
```

▼ Code

```
# Make predictions
logit_pred_prob <- predict(logit_model, test_no_duration, type = "response")
logit_pred_class <- ifelse(logit_pred_prob > 0.5, 1, 0)
```

Significant Variables

▼ Code

```
# Extract coefficients
coef_summary <- summary(logit_model)$coefficients
significant_vars <- coef_summary[coef_summary[, 4] < 0.05, ]

cat("Significant Variables (p < 0.05):\n")
```

```
Significant Variables (p < 0.05):
```

▼ Code

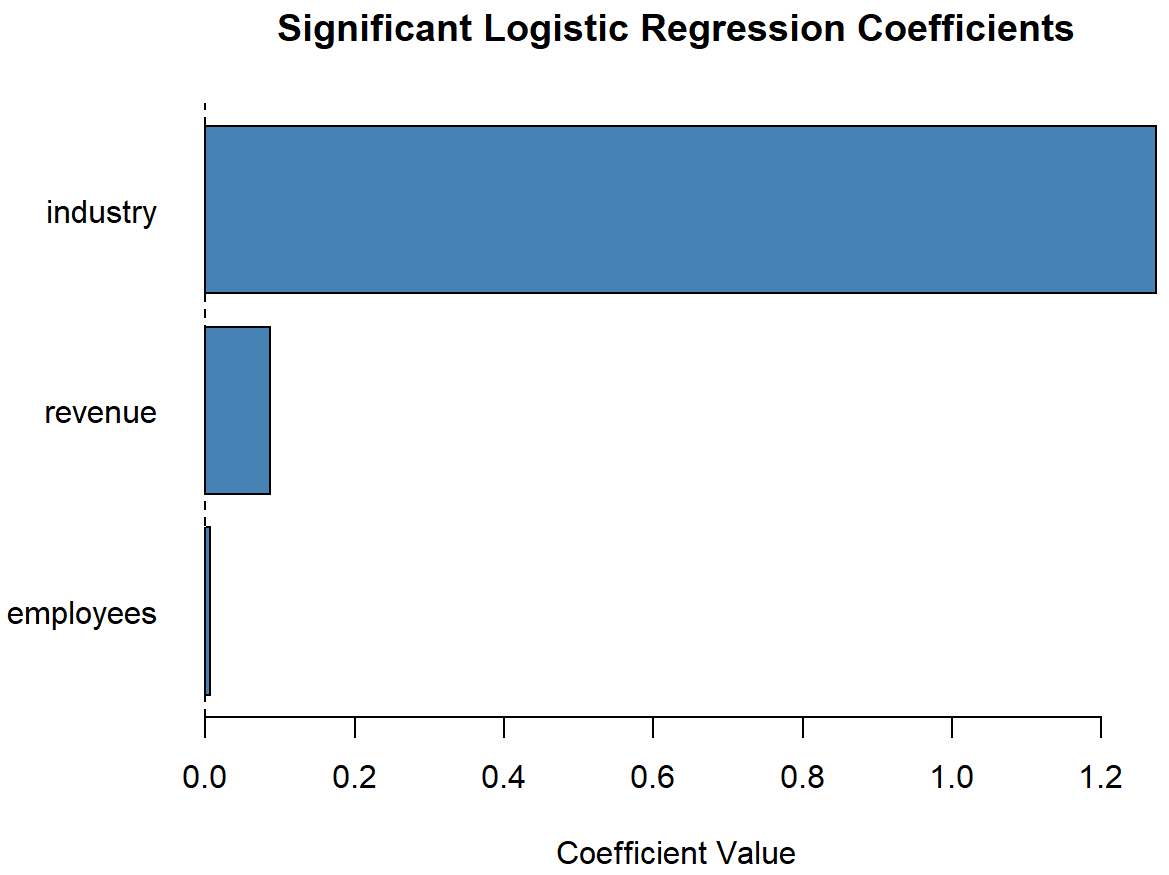
```
print(significant_vars)
```

```
               Estimate   Std. Error   z value     Pr(>|z|)
(Intercept) -7.687180040 1.0653048069 -7.215944 5.356102e-13
```

```
industry     1.272721705 0.3073513643  4.140934 3.458945e-05
revenue      0.086995954 0.0161775551  5.377571 7.549738e-08
employees    0.006690118 0.0008758244  7.638652 2.195082e-14
```

▼ Code

```r
# Plot coefficients
if(nrow(significant_vars) > 1) {
  sig_coefs <- significant_vars[-1, 1]  # Remove intercept

  # Sort by absolute value
  sig_coefs_sorted <- sort(sig_coefs)

  par(mar = c(5, 8, 4, 2))
  barplot(sig_coefs_sorted,
          horiz = TRUE,
          las = 1,
          col = ifelse(sig_coefs_sorted > 0, "steelblue", "coral"),
          main = "Significant Logistic Regression Coefficients",
          xlab = "Coefficient Value")
  abline(v = 0, lty = 2)
  par(mar = c(5, 4, 4, 2))
}
```



Model Comparison

▼ Code

```
# Calculate accuracy
calc_accuracy <- function(actual, predicted) {
  sum(actual == predicted) / length(actual)
}

# Sanity check for data leakage
cat("Diagnostic Check:\n")
```

```
Diagnostic Check:
```

▼ Code

```
cat("Test set size:", nrow(test), "\n")
```

```
Test set size: 150
```

▼ Code

```
cat("Actual acquisition distribution:", table(test$acquisition), "\n")
```

```
Actual acquisition distribution: 57 93
```

▼ Code

```
cat("RF predictions distribution:", table(rf_pred_numeric), "\n\n")
```

```
RF predictions distribution: 40 110
```

▼ Code

```
# Calculate metrics
rf_accuracy <- calc_accuracy(test$acquisition, rf_pred_numeric)
tree_accuracy <- calc_accuracy(test$acquisition, tree_pred_class)
logit_accuracy <- calc_accuracy(test$acquisition, logit_pred_class)

cat("Accuracy Results:\n")
```

```
Accuracy Results:
```

▼ Code

```
cat("Random Forest:", round(rf_accuracy, 4), "\n")
```

```
Random Forest: 0.7667
```

▼ Code

```
cat("Decision Tree:", round(tree_accuracy, 4), "\n")
```

```
Decision Tree: 0.7333
```

▼ Code

```
cat("Logistic Regression:", round(logit_accuracy, 4), "\n\n")
```

```
Logistic Regression: 0.8
```

▼ Code

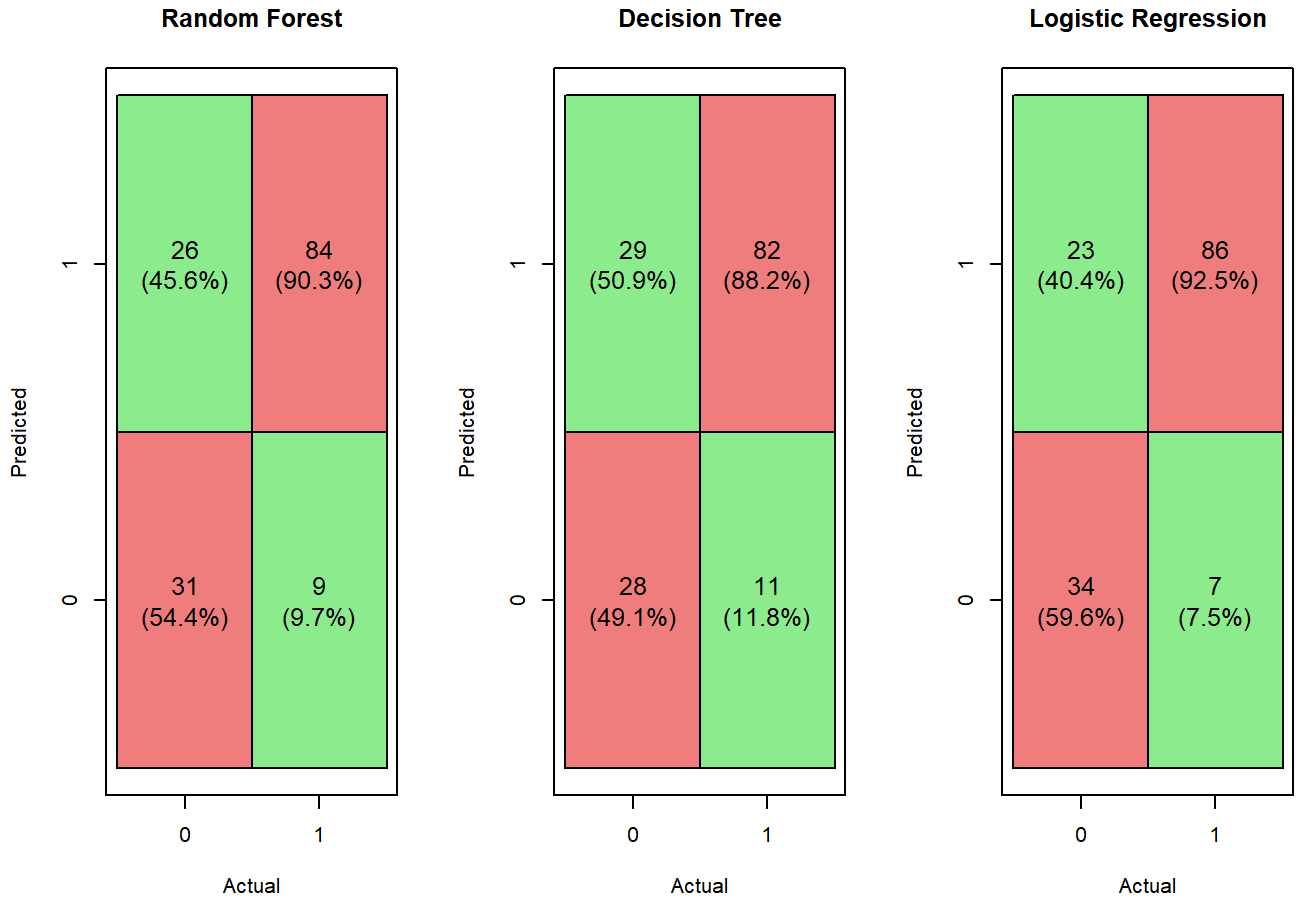
```
if(max(rf_accuracy, tree_accuracy, logit_accuracy) > 0.95) {
  cat("Note: Very high accuracy detected. This could indicate:\n")
  cat("- Strong predictive patterns in the data\n")
  cat("- Potential overfitting (validate on new data)\n\n")
}
```

Confusion Matrices

▼ Code

```
par(mfrow = c(1, 3))

# Function to create nice confusion matrix
plot_confusion <- function(actual, predicted, title) {
  cm <- table(Predicted = predicted, Actual = actual)

  # Calculate percentages
  cm_pct <- prop.table(cm, 2) * 100

  # Create color matrix
  colors <- matrix(c("lightcoral", "lightgreen", "lightgreen", "lightcoral"),
                   nrow = 2)

  # Plot
  plot(0, 0, type = "n", xlim = c(0, 2), ylim = c(0, 2),
       xlab = "Actual", ylab = "Predicted", main = title,
       xaxt = "n", yaxt = "n")

  # Add colored rectangles
  rect(0, 0, 1, 1, col = colors[1, 1], border = "black")
  rect(1, 0, 2, 1, col = colors[1, 2], border = "black")
```

```
  rect(0, 1, 1, 2, col = colors[2, 1], border = "black")
  rect(1, 1, 2, 2, col = colors[2, 2], border = "black")

  # Add text
  text(0.5, 1.5, paste0(cm[2, 1], "\n(", round(cm_pct[2, 1], 1), "%)"), cex = 1.2)
  text(1.5, 1.5, paste0(cm[2, 2], "\n(", round(cm_pct[2, 2], 1), "%)"), cex = 1.2)
  text(0.5, 0.5, paste0(cm[1, 1], "\n(", round(cm_pct[1, 1], 1), "%)"), cex = 1.2)
  text(1.5, 0.5, paste0(cm[1, 2], "\n(", round(cm_pct[1, 2], 1), "%)"), cex = 1.2)

  # Add axis labels
  axis(1, at = c(0.5, 1.5), labels = c("0", "1"))
  axis(2, at = c(0.5, 1.5), labels = c("0", "1"))
}

plot_confusion(test$acquisition, rf_pred_numeric, "Random Forest")
plot_confusion(test$acquisition, tree_pred_class, "Decision Tree")
plot_confusion(test$acquisition, logit_pred_class, "Logistic Regression")
```



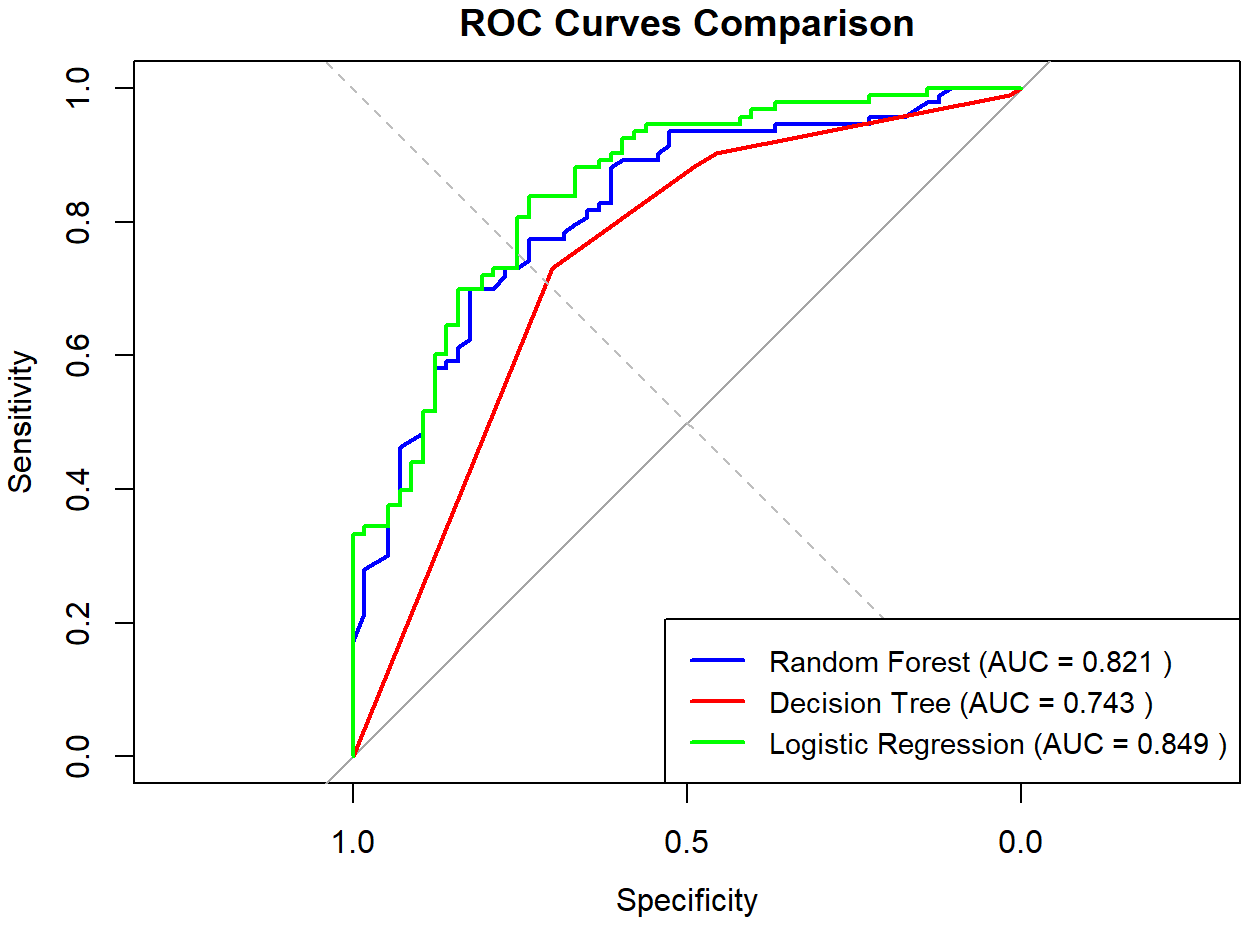
▼ Code

```
par(mfrow = c(1, 1))
```

ROC Curves

▼ Code

```
# Calculate ROC curves
rf_roc <- roc(test$acquisition, rf_pred_prob, quiet = TRUE)
tree_roc <- roc(test$acquisition, tree_pred_prob, quiet = TRUE)
logit_roc <- roc(test$acquisition, logit_pred_prob, quiet = TRUE)

# Plot
plot(rf_roc, col = "blue", lwd = 2, main = "ROC Curves Comparison")
plot(tree_roc, col = "red", lwd = 2, add = TRUE)
plot(logit_roc, col = "green", lwd = 2, add = TRUE)
abline(a = 0, b = 1, lty = 2, col = "gray")

legend("bottomright",
       legend = c(
         paste("Random Forest (AUC =", round(auc(rf_roc), 3), ")"),
         paste("Decision Tree (AUC =", round(auc(tree_roc), 3), ")"),
         paste("Logistic Regression (AUC =", round(auc(logit_roc), 3), ")")
       ),
       col = c("blue", "red", "green"),
       lwd = 2,
       cex = 0.9)
```



Performance Summary Table

▼ Code

```
# Create summary table
summary_df <- data.frame(
  Model = c("Random Forest", "Decision Tree", "Logistic Regression"),
  Accuracy = c(rf_accuracy, tree_accuracy, logit_accuracy),
  AUC = c(auc(rf_roc), auc(tree_roc), auc(logit_roc))
)

print(summary_df)
```

```
                Model  Accuracy       AUC
1       Random Forest 0.7666667 0.8206942
2       Decision Tree 0.7333333 0.7425957
3 Logistic Regression 0.8000000 0.8490851
```

▼ Code

```
best_model <- summary_df$Model[which.max(summary_df$AUC)]
```

```
cat("\nBest Model:", best_model, "\n")
```

```
Best Model: Logistic Regression
```

Duration Prediction

▼ Code

```
# For duration prediction, we use the full dataset (including retention variables)
# because duration is only predicted after acquisition

# Get acquired customers from original data
train_acquired_full <- acquisitionRetention[train_indices, ]
train_acquired_full <- train_acquired_full[train_acquired_full$acquisition == 1, ]

test_acquired_full <- acquisitionRetention[-train_indices, ]
test_acquired_full <- test_acquired_full[test_acquired_full$acquisition == 1, ]

cat("Acquired customers in training:", nrow(train_acquired_full), "\n")
```

```
Acquired customers in training: 245
```

▼ Code

```
cat("Acquired customers in test:", nrow(test_acquired_full), "\n\n")
```

```
Acquired customers in test: 93
```

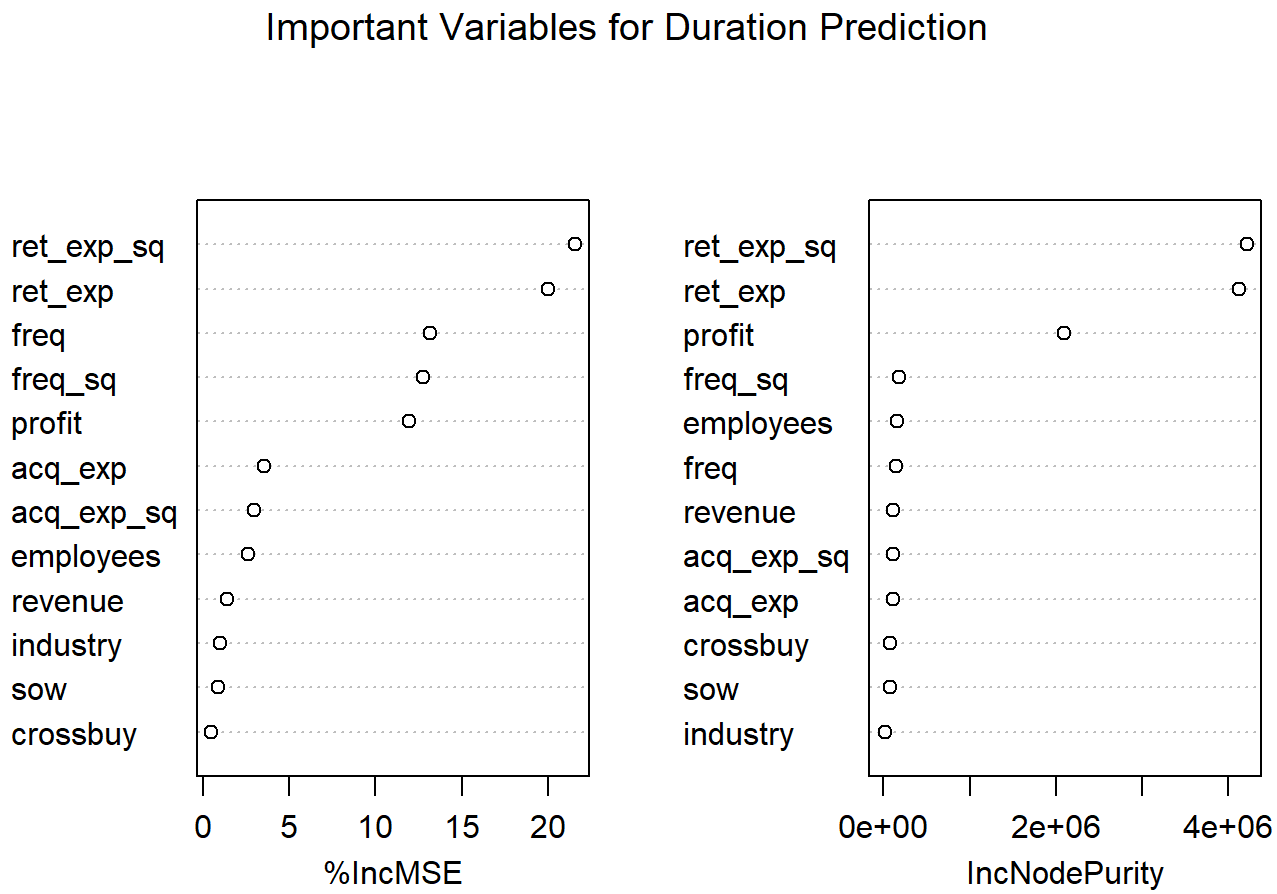
▼ Code

```
if(nrow(train_acquired_full) > 10) {
  # Remove acquisition column and any ID columns for duration model
  duration_predictors <- setdiff(names(train_acquired_full),
                                 c("acquisition", "duration", id_cols))

  cat("Using these predictors for duration:\n")
  print(duration_predictors)
  cat("\n")

  # Build duration model with all available features
  rf_duration <- randomForest(
    duration ~ .,
    data = train_acquired_full[, c("duration", duration_predictors)],
    ntree = 300,
    importance = TRUE
  )
```

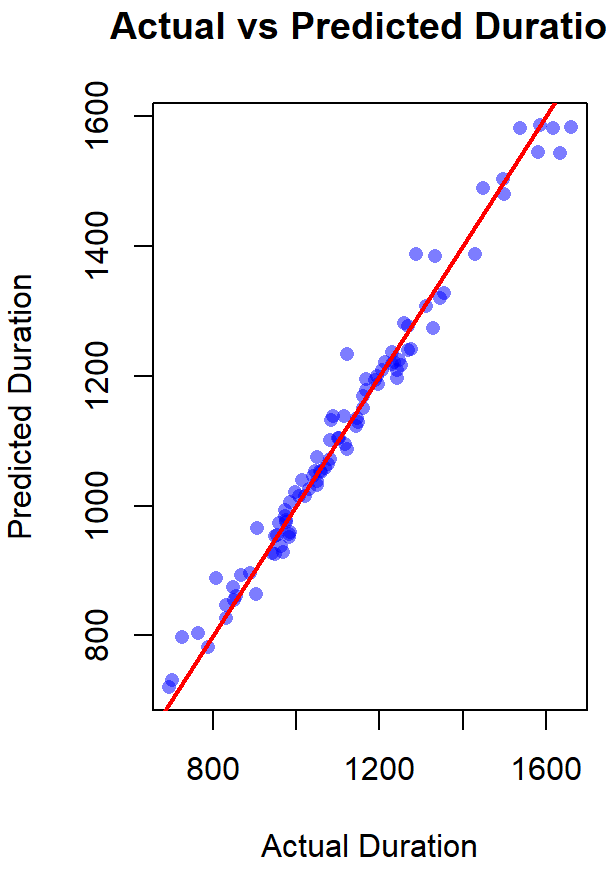
```
  print(rf_duration)

  # Variable importance for duration
  varImpPlot(rf_duration,
             main = "Important Variables for Duration Prediction",
             n.var = min(15, nrow(importance(rf_duration))))

  # Predictions
  if(nrow(test_acquired_full) > 0) {
    duration_pred <- predict(rf_duration, test_acquired_full)

    # Performance metrics
    rmse <- sqrt(mean((test_acquired_full$duration - duration_pred)^2))
    mae <- mean(abs(test_acquired_full$duration - duration_pred))
    r_squared <- cor(test_acquired_full$duration, duration_pred)^2

    cat("\nDuration Model Performance:\n")
    cat("RMSE:", round(rmse, 3), "\n")
    cat("MAE:", round(mae, 3), "\n")
    cat("R-squared:", round(r_squared, 3), "\n")

    # Plot actual vs predicted
    par(mfrow = c(1, 2))

    plot(test_acquired_full$duration, duration_pred,
         xlab = "Actual Duration",
         ylab = "Predicted Duration",
         main = "Actual vs Predicted Duration",
         pch = 16,
         col = rgb(0, 0, 1, 0.5))
    abline(0, 1, col = "red", lwd = 2)

    # Residuals plot
    residuals <- test_acquired_full$duration - duration_pred
    plot(duration_pred, residuals,
         xlab = "Predicted Duration",
         ylab = "Residuals",
         main = "Residual Plot",
         pch = 16,
         col = rgb(1, 0, 0, 0.5))
    abline(h = 0, col = "blue", lwd = 2, lty = 2)

    par(mfrow = c(1, 1))
  }
}
```

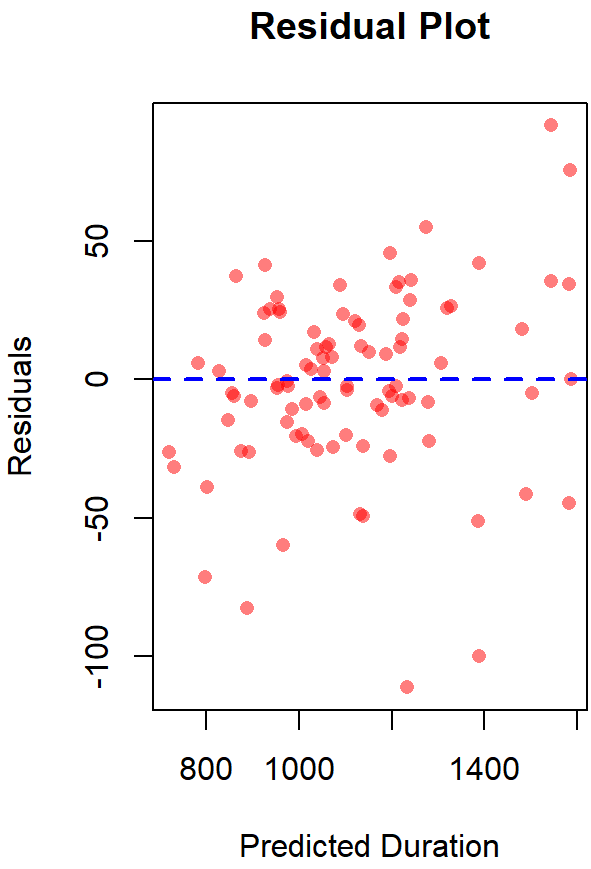
```
Using these predictors for duration:
 [1] "profit"     "acq_exp"    "ret_exp"    "acq_exp_sq" "ret_exp_sq"
 [6] "freq"       "freq_sq"    "crossbuy"   "sow"        "industry"
[11] "revenue"    "employees"
```

```
Call:
 randomForest(formula = duration ~ ., data = train_acquired_full[,      c("duration",
duration_predictors)], ntree = 300, importance = TRUE)
               Type of random forest: regression
                     Number of trees: 300
No. of variables tried at each split: 4

          Mean of squared residuals: 1489.109
                    % Var explained: 96.83
```




```
Duration Model Performance:
RMSE: 32.878
MAE: 24.195
R-squared: 0.978
```





▼ Code

```
# Get importance scores
importance_scores <- importance(rf_duration)
print(importance_scores)
```

```
             %IncMSE IncNodePurity
profit     11.9160609    2094958.36
acq_exp     3.5378055     109569.96
ret_exp    19.9865764    4121608.24
acq_exp_sq  2.9413500     111315.55
ret_exp_sq 21.5385244    4215158.77
freq       13.1692561     149985.52
freq_sq    12.7782403     180242.94
crossbuy    0.4913718      72877.19
sow         0.8812465      71679.23
industry    1.0095582      13874.29
revenue     1.3968278     116799.95
employees   2.6180476     155974.15
```

▼ Code

```
# Sort by %IncMSE
```

```
importance_scores[order(importance_scores[,1], decreasing = TRUE), ]
```

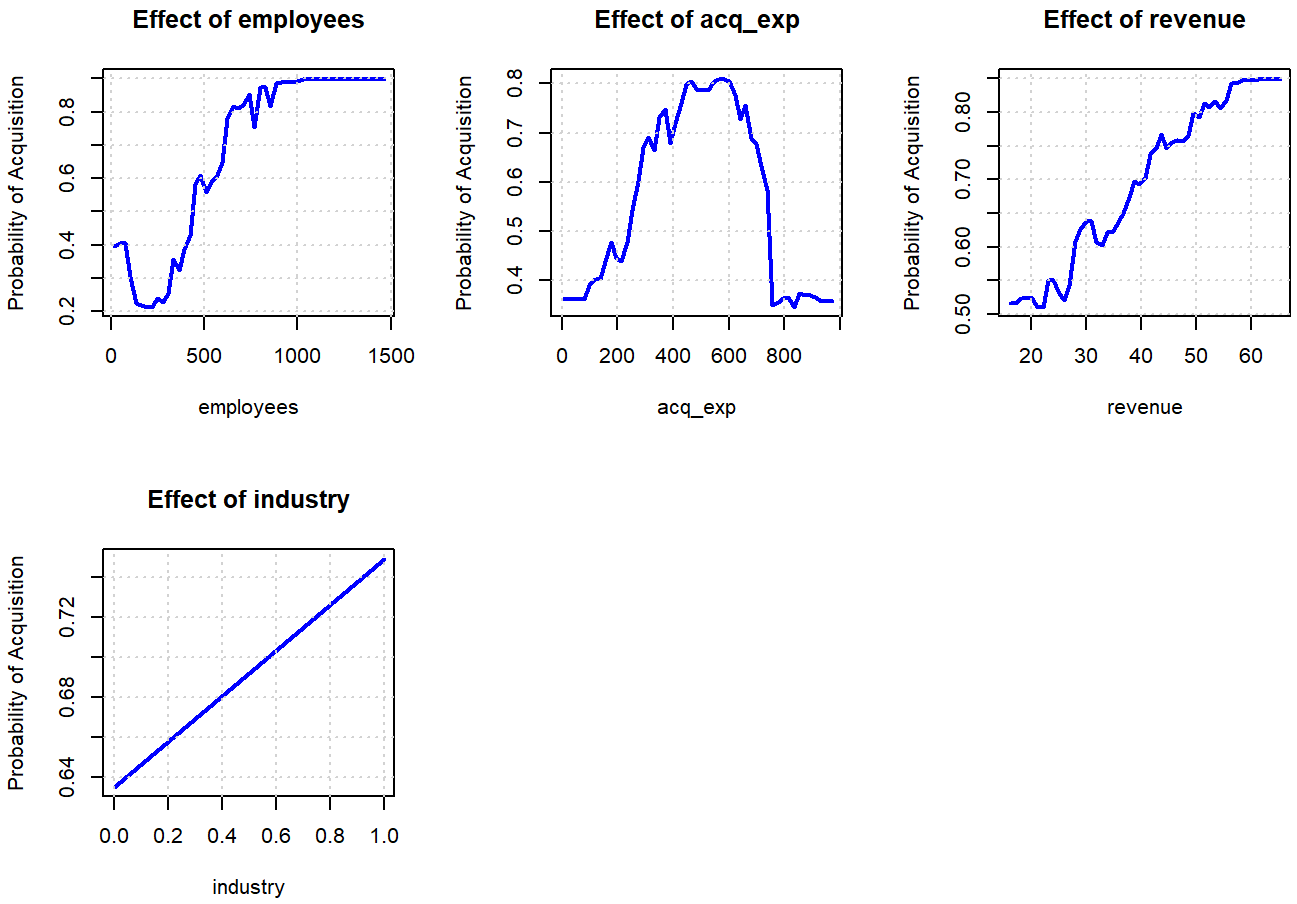
```
             %IncMSE IncNodePurity
ret_exp_sq 21.5385244    4215158.77
ret_exp    19.9865764    4121608.24
freq       13.1692561     149985.52
freq_sq    12.7782403     180242.94
profit     11.9160609    2094958.36
acq_exp     3.5378055     109569.96
acq_exp_sq  2.9413500     111315.55
employees   2.6180476     155974.15
revenue     1.3968278     116799.95
industry    1.0095582      13874.29
sow         0.8812465      71679.23
crossbuy    0.4913718      72877.19
```

Partial Dependence Plots

▼ Code

```
# Get top important variables
imp <- importance(rf_final)
top_vars <- head(rownames(imp)[order(-imp[, 1])], 6)

# Store PDP data for interpretation
pdp_summary <- list()

par(mfrow = c(2, 3))
for(var in top_vars) {
  pdp_data <- partial(rf_final,
                      pred.var = var,
                      prob = TRUE,
                      which.class = "1")

  # Store summary statistics
  pdp_summary[[var]] <- data.frame(
    variable = var,
    min_value = min(pdp_data[[1]]),
    max_value = max(pdp_data[[1]]),
    min_prob = min(pdp_data$yhat),
    max_prob = max(pdp_data$yhat),
    range_effect = max(pdp_data$yhat) - min(pdp_data$yhat),
    mean_prob = mean(pdp_data$yhat)
  )

  # Plot
  plot(pdp_data[[1]], pdp_data$yhat,
       type = "l",
       lwd = 2,
```

```
        col = "blue",
        main = paste("Effect of", var),
        xlab = var,
        ylab = "Probability of Acquisition")
  grid()
}
par(mfrow = c(1, 1))
```



▼ Code

```
# Print summary for interpretation
pdp_df <- do.call(rbind, pdp_summary)
print(pdp_df)
```

```
           variable min_value max_value  min_prob  max_prob range_effect
employees employees     18.00   1461.00 0.2121143 0.8997714    0.6876571
acq_exp     acq_exp      1.21    970.31 0.3454095 0.8100476    0.4646381
revenue     revenue     16.29     65.10 0.5105524 0.8494381    0.3388857
industry   industry      0.00      1.00 0.6350000 0.7494381    0.1144381
           mean_prob
employees 0.6747970
acq_exp   0.5580472
revenue   0.6961122
industry  0.6922190
```

Key Findings

Model Performance

▼ Code

```
cat("Final Model Comparison:\n\n")
```

```
Final Model Comparison:
```

▼ Code

```
print(summary_df)
```

```
                Model  Accuracy       AUC
1       Random Forest 0.7666667 0.8206942
2       Decision Tree 0.7333333 0.7425957
3 Logistic Regression 0.8000000 0.8490851
```

▼ Code

```
cat("\nBest performing model:", best_model, "\n")
```

```

Best performing model: Logistic Regression
```

▼ Code

```
cat("Best AUC:", round(max(summary_df$AUC), 4), "\n")
```

```
Best AUC: 0.8491
```

▼ Code

```
cat("Best Accuracy:", round(max(summary_df$Accuracy), 4), "\n")
```

```
Best Accuracy: 0.8
```

Important Insights

1. The best model was Logistic Regression - Best AUC: 0.8491 Best Accuracy: 0.8
2. The top most important variables for predicting acquisition are: employees, acq_exp, revenue, and industry

▼ Code

```
print(head(importance_df, 5))
```

```
           Variable Importance
employees employees  61.512949
acq_exp     acq_exp  41.862147
revenue     revenue  36.208863
industry   industry   6.749492
```

3. Duration model performance: Random Forest RMSE: 32.878 MAE: 24.195 R-squared: 0.978