Bank Marketing Case Study

Collin Real, Leonel Salazar, Seth Harris, Joaquin Ramirez

```
library(tidyverse)

Warning: package 'ggplot2' was built under R version 4.3.3

Warning: package 'tidyr' was built under R version 4.3.3

Warning: package 'dplyr' was built under R version 4.3.3

Warning: package 'stringr' was built under R version 4.3.2

Warning: package 'lubridate' was built under R version 4.3.2

── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✔ dplyr     1.1.4     ✔ readr     2.1.4
✔ forcats   1.0.0     ✔ stringr   1.5.1
✔ ggplot2   3.5.1     ✔ tibble    3.2.1
✔ lubridate 1.9.3     ✔ tidyr     1.3.1
✔ purrr     1.0.2
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors

library(ggplot2)
library(here)

Warning: package 'here' was built under R version 4.3.3

here() starts at C:/Users/Leonel/Desktop/MSDA/MS Data Analytics/Current Class/DA 6813/Week 3/Case Study Banking

library(caret)

Warning: package 'caret' was built under R version 4.3.3

Loading required package: lattice

Warning: package 'lattice' was built under R version 4.3.3


Attaching package: 'caret'
```

```
The following object is masked from 'package:purrr':

    lift
```

```r
library(dplyr)
library(stats)
library(car)
```

```
Loading required package: carData

Attaching package: 'car'

The following object is masked from 'package:dplyr':

    recode

The following object is masked from 'package:purrr':

    some
```

```r
library(pscl)
```

```
Warning: package 'pscl' was built under R version 4.3.3
```

```
Classes and Methods for R originally developed in the
Political Science Computational Laboratory
Department of Political Science
Stanford University (2002-2015),
by and under the direction of Simon Jackman.
hurdle and zeroinfl functions by Achim Zeileis.
```

```r
library(flextable)
```

```
Warning: package 'flextable' was built under R version 4.3.3


Attaching package: 'flextable'

The following object is masked from 'package:purrr':

    compose
```

```r
library(corrplot)
```

```
Warning: package 'corrplot' was built under R version 4.3.3
```

```
corrplot 0.94 loaded
```

```r
library(pROC)
```

```
Warning: package 'pROC' was built under R version 4.3.3
```

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

Attaching package: 'pROC'

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

    cov, smooth, var
```

```r
library(randomForest)
```

```
Warning: package 'randomForest' was built under R version 4.3.3

randomForest 4.7-1.1
Type rfNews() to see new features/changes/bug fixes.

Attaching package: 'randomForest'

The following object is masked from 'package:dplyr':

    combine

The following object is masked from 'package:ggplot2':

    margin
```

```r
# Use the here function to construct the file path and import the dataset
data <- read.csv(here("bank-additional_clean.csv"), header = TRUE, sep = ",")

# View the first few rows of the dataset
head(data)
```

```
  age         job marital         education default housing    loan   contact
1  30 blue-collar married          basic.9y      no     yes      no  cellular
2  39    services  single       high.school      no      no      no telephone
3  25    services married       high.school      no     yes      no telephone
4  38    services married          basic.9y      no unknown unknown telephone
5  47      admin. married university.degree      no     yes      no  cellular
6  32    services  single university.degree      no      no      no  cellular
  month day_of_week duration campaign pdays previous    poutcome emp.var.rate
1   may         fri      487        2   999        0 nonexistent         -1.8
2   may         fri      346        4   999        0 nonexistent          1.1
3   jun         wed      227        1   999        0 nonexistent          1.4
4   jun         fri       17        3   999        0 nonexistent          1.4
5   nov         mon       58        1   999        0 nonexistent         -0.1
6   sep         thu      128        3   999        2     failure         -1.1
  cons.price.idx cons.conf.idx euribor3m nr.employed  y
1         92.893         -46.2     1.313      5099.1 no
2         93.994         -36.4     4.855      5191.0 no
3         94.465         -41.8     4.962      5228.1 no
4         94.465         -41.8     4.959      5228.1 no
```

```
5         93.200         -42.0     4.191      5195.8 no
6         94.199         -37.5     0.884      4963.6 no

str(data)

'data.frame':   4119 obs. of  21 variables:
 $ age           : int  30 39 25 38 47 32 32 41 31 35 ...
 $ job           : chr  "blue-collar" "services" "services" "services" ...
 $ marital       : chr  "married" "single" "married" "married" ...
 $ education     : chr  "basic.9y" "high.school" "high.school" "basic.9y" ...
 $ default       : chr  "no" "no" "no" "no" ...
 $ housing       : chr  "yes" "no" "yes" "unknown" ...
 $ loan          : chr  "no" "no" "no" "unknown" ...
 $ contact       : chr  "cellular" "telephone" "telephone" "telephone" ...
 $ month         : chr  "may" "may" "jun" "jun" ...
 $ day_of_week   : chr  "fri" "fri" "wed" "fri" ...
 $ duration      : int  487 346 227 17 58 128 290 44 68 170 ...
 $ campaign      : int  2 4 1 3 1 3 4 2 1 1 ...
 $ pdays         : int  999 999 999 999 999 999 999 999 999 999 ...
 $ previous      : int  0 0 0 0 0 2 0 0 1 0 ...
 $ poutcome      : chr  "nonexistent" "nonexistent" "nonexistent"
"nonexistent" ...
 $ emp.var.rate  : num  -1.8 1.1 1.4 1.4 -0.1 -1.1 -1.1 -0.1 -0.1 1.1 ...
 $ cons.price.idx: num  92.9 94 94.5 94.5 93.2 ...
 $ cons.conf.idx : num  -46.2 -36.4 -41.8 -41.8 -42 -37.5 -37.5 -42 -42 -36.4
...
 $ euribor3m     : num  1.31 4.86 4.96 4.96 4.19 ...
 $ nr.employed   : num  5099 5191 5228 5228 5196 ...
 $ y             : chr  "no" "no" "no" "no" ...

# Function to check for missing (NA) values and "unknown" entries in all
columns
check_missing_unknown <- function(data) {
  result <- data.frame(
    Variable = colnames(data),
    Missing_Count = sapply(data, function(x) sum(is.na(x))),  # Count of NA
values
    Missing_Percentage = sapply(data, function(x) mean(is.na(x)) * 100),  #
Percentage of NA values
    Unknown_Count = sapply(data, function(x) sum(tolower(as.character(x)) ==
"unknown", na.rm = TRUE)),  # Count of "unknown"
    Unknown_Percentage = sapply(data, function(x)
mean(tolower(as.character(x)) == "unknown", na.rm = TRUE) * 100)  #
Percentage of "unknown"
  )
  return(result)
}



missing_unknown_summary <- check_missing_unknown(data)
```

```
# Display the summary table
print(missing_unknown_summary)
```

```
                       Variable Missing_Count Missing_Percentage Unknown_Count
age                         age             0                  0             0
job                         job             0                  0            39
marital                 marital             0                  0            11
education             education             0                  0           167
default                 default             0                  0           803
housing                 housing             0                  0           105
loan                       loan             0                  0           105
contact                 contact             0                  0             0
month                     month             0                  0             0
day_of_week         day_of_week             0                  0             0
duration               duration             0                  0             0
campaign               campaign             0                  0             0
pdays                     pdays             0                  0             0
previous               previous             0                  0             0
poutcome               poutcome             0                  0             0
emp.var.rate       emp.var.rate             0                  0             0
cons.price.idx cons.price.idx             0                  0             0
cons.conf.idx     cons.conf.idx             0                  0             0
euribor3m             euribor3m             0                  0             0
nr.employed         nr.employed             0                  0             0
y                             y             0                  0             0
               Unknown_Percentage
age                     0.0000000
job                     0.9468318
marital                 0.2670551
education               4.0543821
default                19.4950231
housing                 2.5491624
loan                    2.5491624
contact                 0.0000000
month                   0.0000000
day_of_week             0.0000000
duration                0.0000000
campaign                0.0000000
pdays                   0.0000000
previous                0.0000000
poutcome                0.0000000
emp.var.rate            0.0000000
cons.price.idx          0.0000000
cons.conf.idx           0.0000000
euribor3m               0.0000000
nr.employed             0.0000000
y                       0.0000000
```

```
# Convert all integer and numeric variables to numeric type
data[] <- lapply(data, function(x) {
  if (is.integer(x) || is.numeric(x)) {
    return(as.numeric(x))  # Convert to numeric
  } else if (is.character(x)) {
    return(factor(x))  # Convert character to factor
  } else {
    return(x)  # Leave other types unchanged
  }
})

# Verify the changes
str(data)

'data.frame':   4119 obs. of  21 variables:
 $ age           : num  30 39 25 38 47 32 32 41 31 35 ...
 $ job           : Factor w/ 12 levels "admin.","blue-collar",..: 2 8 8 8 1 8
1 3 8 2 ...
 $ marital       : Factor w/ 4 levels "divorced","married",..: 2 3 2 2 2 3 3
2 1 2 ...
 $ education     : Factor w/ 8 levels "basic.4y","basic.6y",..: 3 4 4 3 7 7 7
7 6 3 ...
 $ default       : Factor w/ 3 levels "no","unknown",..: 1 1 1 1 1 1 1 2 1 2
...
 $ housing       : Factor w/ 3 levels "no","unknown",..: 3 1 3 2 3 1 3 3 1 1
...
 $ loan          : Factor w/ 3 levels "no","unknown",..: 1 1 1 2 1 1 1 1 1 1
...
 $ contact       : Factor w/ 2 levels "cellular","telephone": 1 2 2 2 1 1 1 1
1 2 ...
 $ month         : Factor w/ 10 levels "apr","aug","dec",..: 7 7 5 5 8 10 10
8 8 7 ...
 $ day_of_week   : Factor w/ 5 levels "fri","mon","thu",..: 1 1 5 1 2 3 2 2 4
3 ...
 $ duration      : num  487 346 227 17 58 128 290 44 68 170 ...
 $ campaign      : num  2 4 1 3 1 3 4 2 1 1 ...
 $ pdays         : num  999 999 999 999 999 999 999 999 999 999 ...
 $ previous      : num  0 0 0 0 0 2 0 0 1 0 ...
 $ poutcome      : Factor w/ 3 levels "failure","nonexistent",..: 2 2 2 2 2 1
2 2 1 2 ...
 $ emp.var.rate  : num  -1.8 1.1 1.4 1.4 -0.1 -1.1 -1.1 -0.1 -0.1 1.1 ...
 $ cons.price.idx: num  92.9 94 94.5 94.5 93.2 ...
 $ cons.conf.idx : num  -46.2 -36.4 -41.8 -41.8 -42 -37.5 -37.5 -42 -42 -36.4
...
 $ euribor3m     : num  1.31 4.86 4.96 4.96 4.19 ...
 $ nr.employed   : num  5099 5191 5228 5228 5196 ...
 $ y             : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...

# Transform the 'pdays' column
data_clean <- data %>%
```

```r
  mutate(pdays = ifelse(pdays == 999, "not previously contacted", "previously contacted"))

# Convert 'pdays' to a factor
data_clean$pdays <- as.factor(data_clean$pdays)

# Display the first few rows to verify the change
head(data_clean)
```

```
  age         job marital         education default housing    loan   contact
1  30 blue-collar married          basic.9y      no     yes      no  cellular
2  39    services  single       high.school      no      no      no telephone
3  25    services married       high.school      no     yes      no telephone
4  38    services married          basic.9y      no unknown unknown telephone
5  47      admin. married university.degree      no     yes      no  cellular
6  32    services  single university.degree      no      no      no  cellular
  month day_of_week duration campaign                    pdays previous
1   may         fri      487        2 not previously contacted        0
2   may         fri      346        4 not previously contacted        0
3   jun         wed      227        1 not previously contacted        0
4   jun         fri       17        3 not previously contacted        0
5   nov         mon       58        1 not previously contacted        0
6   sep         thu      128        3 not previously contacted        2
     poutcome emp.var.rate cons.price.idx cons.conf.idx euribor3m nr.employed
1 nonexistent         -1.8         92.893         -46.2     1.313      5099.1
2 nonexistent          1.1         93.994         -36.4     4.855      5191.0
3 nonexistent          1.4         94.465         -41.8     4.962      5228.1
4 nonexistent          1.4         94.465         -41.8     4.959      5228.1
5 nonexistent         -0.1         93.200         -42.0     4.191      5195.8
6     failure         -1.1         94.199         -37.5     0.884      4963.6
   y
1 no
2 no
3 no
4 no
5 no
6 no
```

```r
# Remove the 'duration' variable
data_clean <- select(data_clean, -duration)

# Save the cleaned dataset as a new CSV file
write.csv(data_clean, "data_clean.csv", row.names = FALSE)

# Verify the dataset has been saved
file.exists("data_clean.csv")
```

```
[1] TRUE
```

```
# Export the dataset to a CSV file in the same location
write_csv(data, here("data_new.csv"))

# Confirm the file is saved by showing the file path
cat("Data has been exported to:", here("data_new.csv"))

Data has been exported to: C:/Users/Leonel/Desktop/MSDA/MS Data
Analytics/Current Class/DA 6813/Week 3/Case Study Banking/data_new.csv

# Check the distribution of the target variable 'y'
table(data$y)


  no  yes
3668  451

# Calculate the percentage distribution of 'y'
prop.table(table(data$y)) * 100


      no      yes
89.05074 10.94926

# Function to count "unknown" values in each column
count_unknowns <- function(df) {
  sapply(df, function(x) sum(x == "unknown", na.rm = TRUE))
}

# Apply the function to the dataset
unknown_counts <- count_unknowns(data)

# Display the counts
print(unknown_counts)

          age           job       marital     education       default
            0            39            11           167           803
      housing          loan       contact         month   day_of_week
          105           105             0             0             0
     duration      campaign         pdays      previous      poutcome
            0             0             0             0             0
  emp.var.rate cons.price.idx  cons.conf.idx     euribor3m   nr.employed
            0             0             0             0             0
            y
            0

# Calculate the total number of "unknown" values across all variables
total_unknowns <- sum(unknown_counts)
print(paste("Total number of 'unknown' values across all variables:",
total_unknowns))

[1] "Total number of 'unknown' values across all variables: 1230"
```

```r
# Check for NA values in each column
na_counts <- sapply(data, function(x) sum(is.na(x)))

# Display the counts of NA values for each column
print(na_counts)
```

```
          age           job       marital     education       default 
            0             0             0             0             0 
      housing          loan       contact         month   day_of_week 
            0             0             0             0             0 
     duration      campaign         pdays      previous      poutcome 
            0             0             0             0             0 
 emp.var.rate cons.price.idx  cons.conf.idx     euribor3m   nr.employed 
            0             0             0             0             0 
            y 
            0 
```

```r
# Calculate the total number of NA values across all columns
total_na <- sum(na_counts)
print(paste("Total number of NA values across all columns:", total_na))
```

```
[1] "Total number of NA values across all columns: 0"
```

```r
# Up-sample the minority class
set.seed(123)  # For reproducibility
data_clean_upsampled <- upSample(x = data_clean %>% select(-y), y =
data_clean$y)  # 'y' is the target variable

# Check the new class distribution
table(data_clean_upsampled$Class)
```

```

  no  yes 
3668 3668 
```

```r
# Down-sample the majority class
set.seed(123)  # For reproducibility
data_clean_downsampled <- downSample(x = data_clean %>% select(-y), y =
data_clean$y)  # 'y' is the target variable

# Check the new class distribution
table(data_clean_downsampled$Class)
```

```

 no yes 
451 451 
```

```r
# Load necessary libraries
library(dplyr)
```

```
# Calculate the count of "unknown" values for each column
unknown_counts <- sapply(data_clean_downsampled, function(x) sum(x ==
"unknown", na.rm = TRUE))

# Convert the counts to a data frame for better visualization
unknown_counts_df <- data.frame(
  Variable = names(unknown_counts),
  Unknown_Count = unknown_counts
)

# Calculate the proportion of "unknown" values for each column
unknown_counts_df <- unknown_counts_df %>%
  mutate(Total_Count = nrow(data_clean_downsampled),  # Total rows in the
dataset
         Unknown_Proportion = Unknown_Count / Total_Count * 100)  #
Proportion in percentage

# Display the data frame with counts and proportions
print(unknown_counts_df)

                      Variable Unknown_Count Total_Count Unknown_Proportion
age                        age             0         902          0.0000000
job                        job             6         902          0.6651885
marital                marital             1         902          0.1108647
education            education            40         902          4.4345898
default                default           141         902         15.6319290
housing                housing            20         902          2.2172949
loan                      loan            20         902          2.2172949
contact                contact             0         902          0.0000000
month                    month             0         902          0.0000000
day_of_week        day_of_week             0         902          0.0000000
campaign              campaign             0         902          0.0000000
pdays                    pdays             0         902          0.0000000
previous              previous             0         902          0.0000000
poutcome              poutcome             0         902          0.0000000
emp.var.rate      emp.var.rate             0         902          0.0000000
cons.price.idx cons.price.idx             0         902          0.0000000
cons.conf.idx    cons.conf.idx             0         902          0.0000000
euribor3m            euribor3m             0         902          0.0000000
nr.employed        nr.employed             0         902          0.0000000
Class                    Class             0         902          0.0000000

# Calculate the overall percentage of "unknown" values across all variables
total_unknowns <- sum(unknown_counts)
total_values <- nrow(data_clean_downsampled) * ncol(data_clean_downsampled)
# Total number of data points
overall_unknown_percentage <- (total_unknowns / total_values) * 100
```

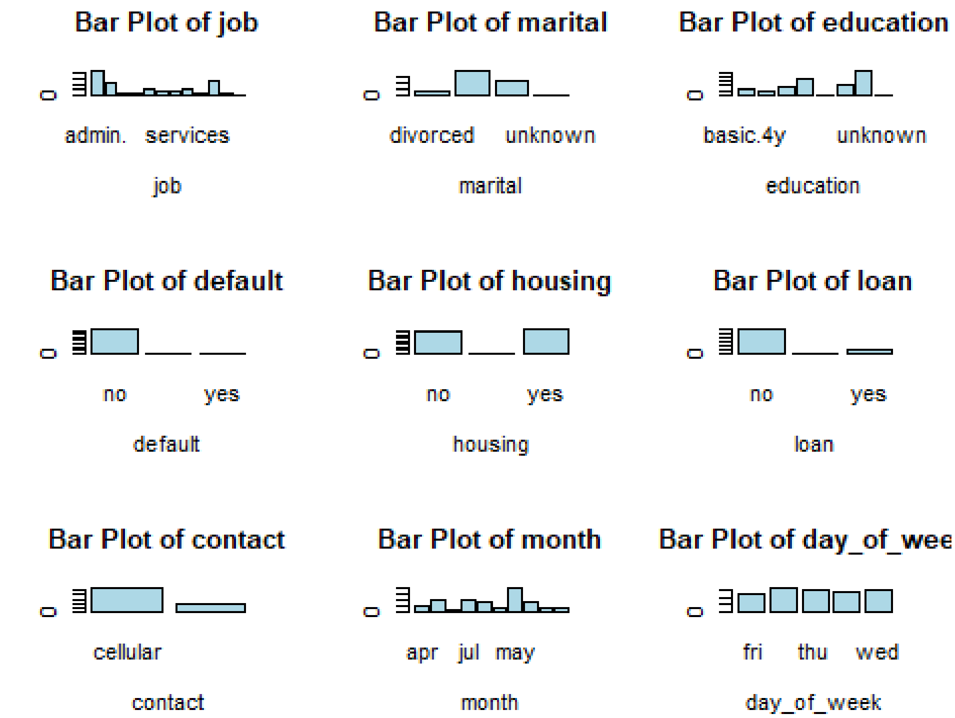
```
print(paste("Overall percentage of 'unknown' values in the dataset:",
round(overall_unknown_percentage, 2), "%"))

[1] "Overall percentage of 'unknown' values in the dataset: 1.26 %"

# Remove rows with 'unknown' values
data_clean_downsampled_no_unknown <- data_clean_downsampled %>%
  filter_all(~ . != "unknown")

# Check the new size of the dataset
print(dim(data_clean_downsampled_no_unknown))

[1] 708  20

# Load necessary libraries
library(dplyr)

# Remove rows with 'unknown' values from the downsampled data
data_clean_downsampled_no_unknown <- data_clean_downsampled %>%
  filter_all(~ . != "unknown")

# Check the distribution of the target variable 'y' after removing unknowns
balance_after_cleaning <- table(data_clean_downsampled_no_unknown$Class)  #
Assuming 'Class' is the name of the target variable column

# Print the class balance
print(balance_after_cleaning)


 no yes
338 370

# Calculate and display the proportion of each class
balance_proportion <- prop.table(balance_after_cleaning) * 100
print(balance_proportion)


      no      yes
47.74011 52.25989

# Export the dataset to a CSV file in the same location
write_csv(data, here("data_clean_downsampled_no_unknown .csv"))

# Confirm the file is saved by showing the file path
cat("Data has been exported to:", here("data_clean_downsampled_no_unknown
.csv"))

Data has been exported to: C:/Users/Leonel/Desktop/MSDA/MS Data
Analytics/Current Class/DA 6813/Week 3/Case Study
Banking/data_clean_downsampled_no_unknown .csv
```

```r
df <- data_clean_downsampled_no_unknown


# Rename the column 'Class' to 'y'
df <- df %>% rename(y = Class)

# Identify categorical and numeric variables
variables <- names(df)
var_types <- ifelse(sapply(df, is.numeric), "Numeric", "Categorical")

# Create a data frame to store variable names and their types
var_table <- data.frame(Variable = variables, Type = var_types)

# Create a flextable
flex_table <- flextable(var_table)

# Apply custom formatting:
# Highlight "Categorical" types with light blue, and "Numeric" types with
light pink
flex_table <- flextable::bg(flex_table, j = "Type", i = ~ Type ==
"Categorical", bg = "lightblue")  # Highlight categorical
flex_table <- flextable::bg(flex_table, j = "Type", i = ~ Type == "Numeric",
bg = "lightpink")      # Highlight numeric

# Adjust column widths for better readability
flex_table <- autofit(flex_table)

# Display the flextable
flex_table
```

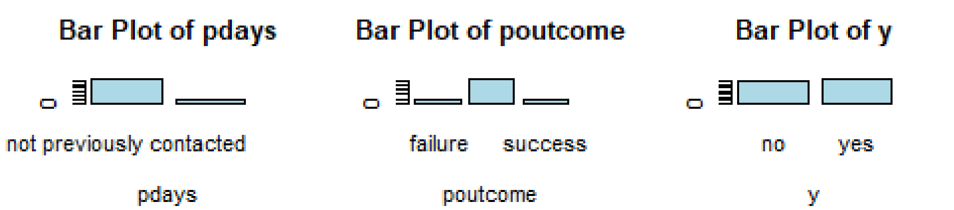
Variable	Type
age	Numeric
job	Categorical
marital	Categorical
education	Categorical
default	Categorical
housing	Categorical
loan	Categorical
contact	Categorical
month	Categorical
day_of_week	Categorical

Variable	Type
campaign	Numeric
pdays	Categorical
previous	Numeric
poutcome	Categorical
emp.var.rate	Numeric
cons.price.idx	Numeric
cons.conf.idx	Numeric
euribor3m	Numeric
nr.employed	Numeric
y	Categorical

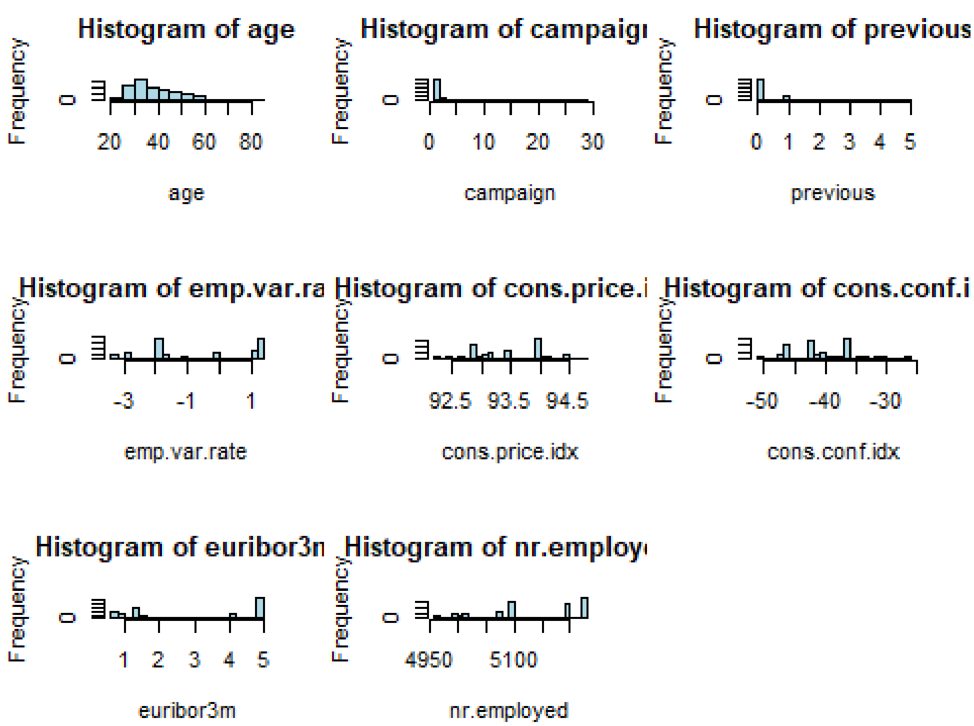
```
# Set up a 3x3 plotting area
par(mfrow=c(3,3))

# Identify categorical variables (factor or character type)
categorical_vars <- sapply(df, function(x) is.factor(x) | is.character(x))
categorical_data <- df[, categorical_vars]  # Subset the dataframe for
categorical variables

# Loop through all categorical variables and plot bar plots
for (var in names(categorical_data)) {
  barplot(table(categorical_data[[var]]), main=paste("Bar Plot of", var),
xlab=var, col="lightblue")
}
```




```
# Reset the plotting layout to 1x1 after plotting
par(mfrow=c(1,1))
```

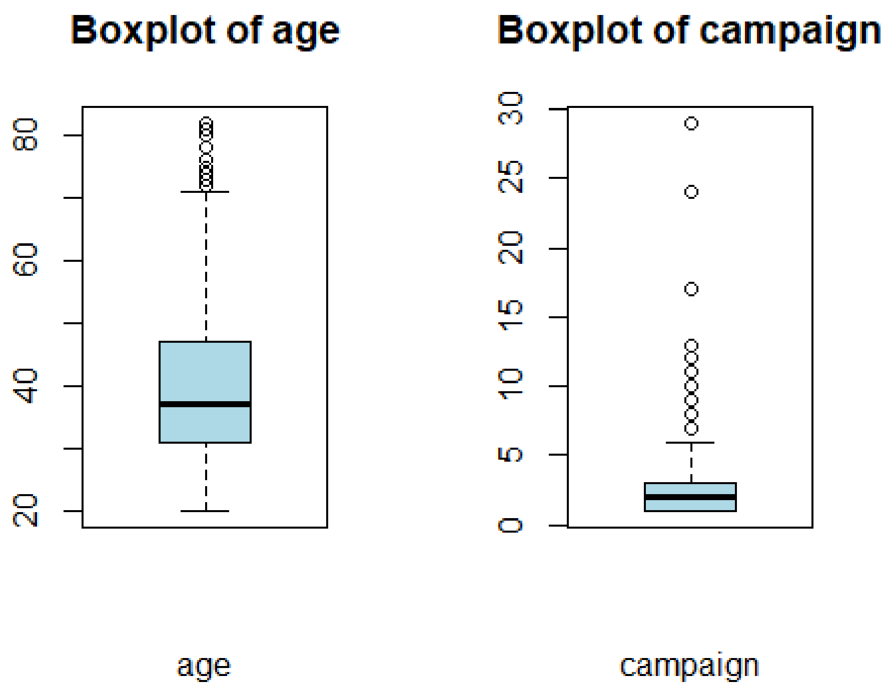


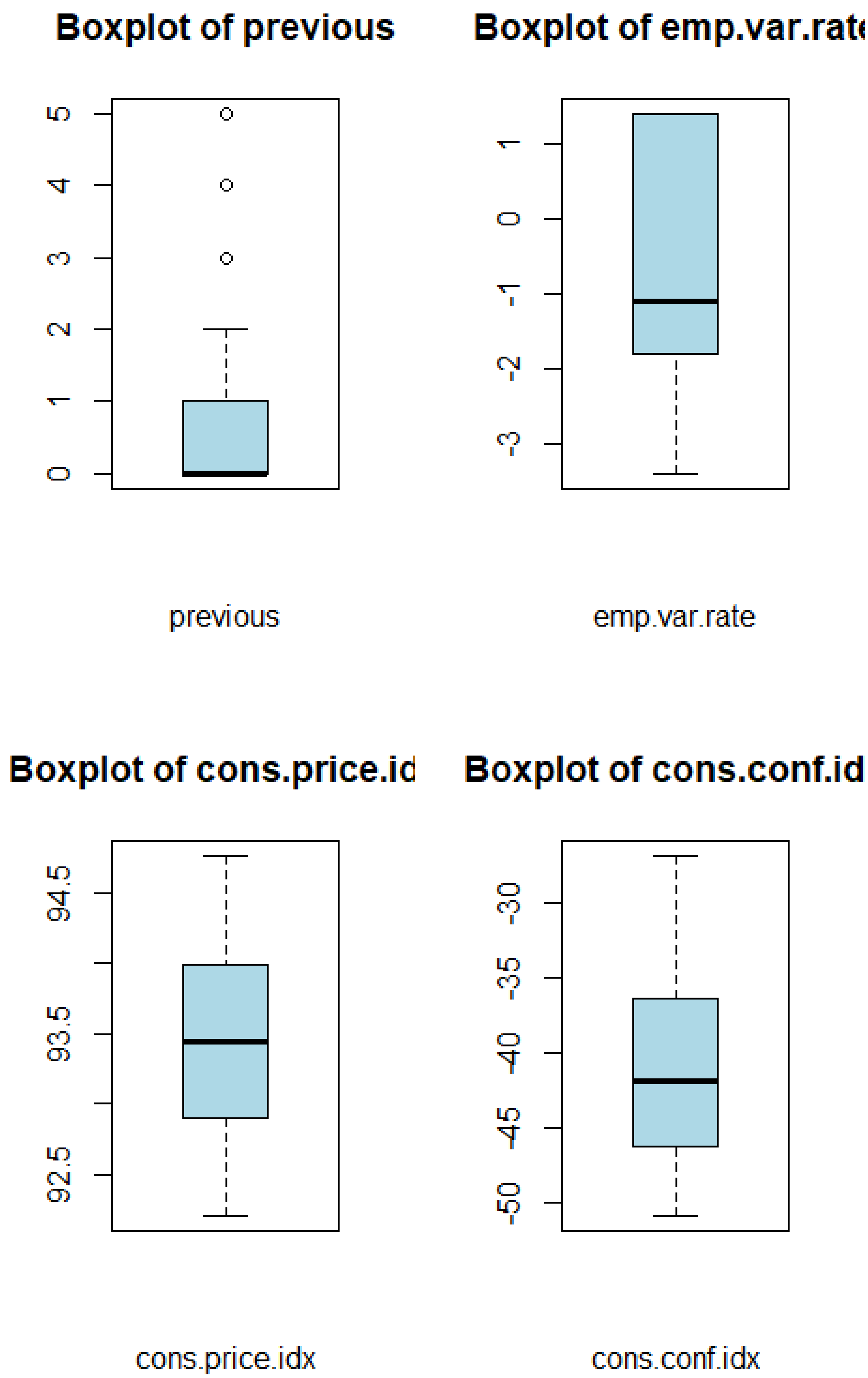
```
# Set up a 3x3 plotting area
par(mfrow=c(3,3))

# Loop through all the columns and plot histograms for continuous (numeric) variables
numeric_vars <- sapply(df, is.numeric)  # Identify numeric variables
continuous_vars <- df[, numeric_vars]   # Subset the dataframe for numeric variables

# Plot histograms for each numeric variable
for (var in names(continuous_vars)) {
  hist(continuous_vars[[var]], main=paste("Histogram of", var), xlab=var, col="lightblue", breaks=20)
}

# Reset the plotting layout to 1x1 after plotting
par(mfrow=c(1,1))
```



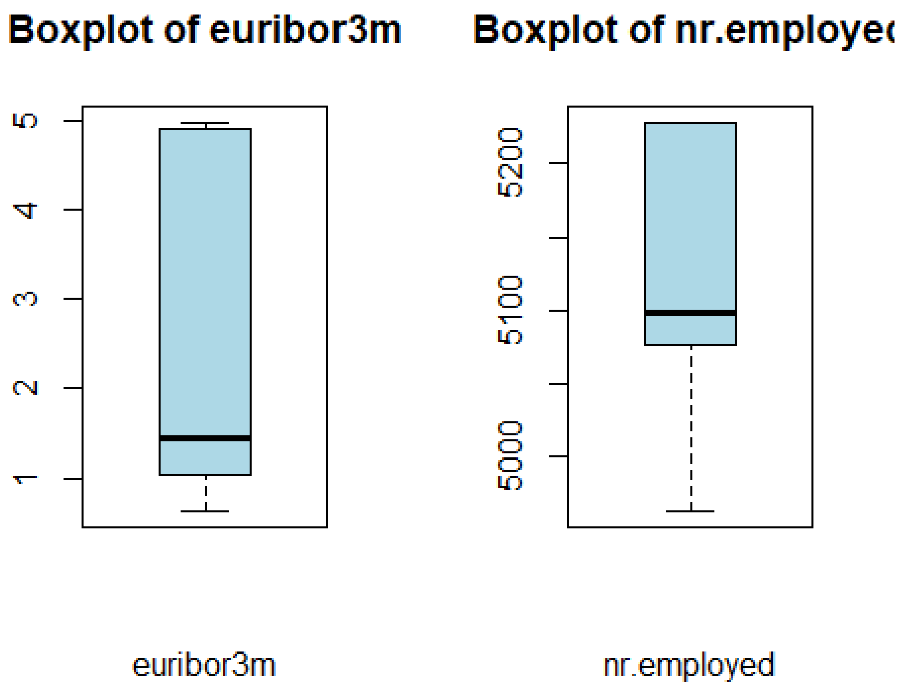

```
# Set up a 3x3 plotting area
par(mfrow=c(1,2))

# Identify numeric variables in the dataset
numeric_vars <- sapply(df, is.numeric)

# Subset the dataset to include only the numeric variables
numeric_data <- df[, numeric_vars]
```

```
# Loop through the numeric variables and create boxplots
for (var in names(numeric_data)) {
  boxplot(numeric_data[[var]], main=paste("Boxplot of", var), xlab=var,
col="lightblue")
}
```

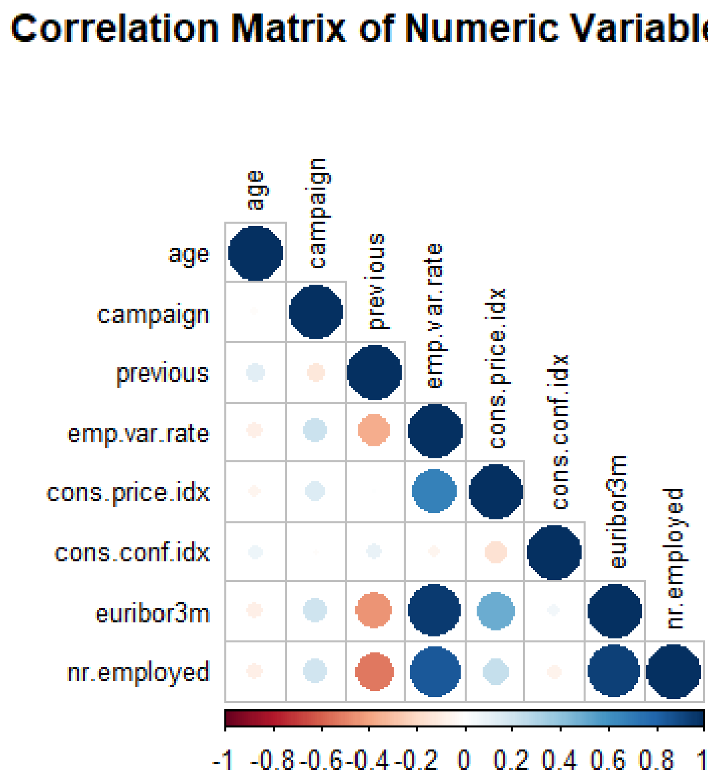




Boxplot of previous
5
4
3
2
1
0
previous
Boxplot of emp.var.rate
1
0
-1
-2
-3
emp.var.rate
Boxplot of cons.price.id
94.5
93.5
92.5
cons.price.idx
Boxplot of cons.conf.id
-30
-35
-40
-45
-50
cons.conf.idx




```r
# Reset the plotting layout to 1x1 after plotting

library(corrplot)

# Select only the numeric variables from the dataset
numeric_vars <- df[, sapply(df, is.numeric)]

# Create the correlation matrix
cor_matrix <- cor(numeric_vars, use="complete.obs")

# Visualize the correlation matrix
corrplot(cor_matrix, method="circle", type="lower",
         tl.col="black", tl.cex=0.8, title="Correlation Matrix of Numeric
Variables",
         mar=c(0,0,1,0))
```



Key Assumptions of Logistic Regression:

1. **Binary Outcome Variable**: The dependent variable should be binary.
2. **Independence of Observations**: Observations should be independent of each other.
3. **No Multicollinearity**: Predictor variables should not be highly correlated with each other.
4. **Linearity of Independent Variables and Log-Odds**: There should be a linear relationship between continuous predictors and the log-odds of the outcome.
5. **Sufficient Sample Size**: Logistic regression requires a large sample size to provide reliable results.

Steps to Test the Assumptions

1. Check for Binary Outcome Variable

Ensure the dependent variable (y) is binary.

```
# Check the levels of the outcome variable
table(df$y)


 no yes
338 370

# Check for variables with only one level
lapply(df, function(x) length(unique(x)))
```

```
$age
[1] 57

$job
[1] 11

$marital
[1] 3

$education
[1] 6

$default
[1] 1

$housing
[1] 2

$loan
[1] 2

$contact
[1] 2

$month
[1] 10

$day_of_week
[1] 5

$campaign
[1] 16

$pdays
[1] 2

$previous
[1] 6

$poutcome
[1] 3

$emp.var.rate
[1] 9

$cons.price.idx
[1] 25
```

```
$cons.conf.idx
[1] 25

$euribor3m
[1] 170

$nr.employed
[1] 10

$y
[1] 2

# Set a seed for reproducibility
set.seed(123)

# Split the data into training (70%) and testing (30%) sets
trainIndex <- createDataPartition(df$y, p = 0.7, list = FALSE)
train_data <- df[trainIndex, ]  # 70% training data
test_data  <- df[-trainIndex, ] # 30% test data

# Fit the logistic regression model on the training set
logit_model_train <- glm(y ~ age + job + marital + education + housing + loan
+ contact + month +
                         day_of_week + campaign + pdays + previous + poutcome
+ emp.var.rate +
                         cons.price.idx + cons.conf.idx,
                         family = binomial, data = train_data)

# Display the summary of the model
summary(logit_model_train)


Call:
glm(formula = y ~ age + job + marital + education + housing +
    loan + contact + month + day_of_week + campaign + pdays +
    previous + poutcome + emp.var.rate + cons.price.idx + cons.conf.idx,
    family = binomial, data = train_data)

Coefficients:
                               Estimate Std. Error z value Pr(>|z|)
(Intercept)                  -1.342e+02  4.230e+01  -3.173  0.00151 **
age                           2.352e-02  1.550e-02   1.517  0.12915
jobblue-collar                8.347e-01  4.557e-01   1.832  0.06701 .
jobentrepreneur               3.124e-01  6.185e-01   0.505  0.61350
jobhousemaid                  1.336e+00  1.152e+00   1.160  0.24621
jobmanagement                -7.618e-02  4.796e-01  -0.159  0.87381
jobretired                    1.304e+00  8.250e-01   1.580  0.11401
jobself-employed             -4.210e-01  6.558e-01  -0.642  0.52093
jobservices                  -5.656e-01  4.996e-01  -1.132  0.25754
jobstudent                   -1.258e-01  8.927e-01  -0.141  0.88795
```

```
jobtechnician                  1.738e-01  4.033e-01   0.431  0.66657    
jobunemployed                  1.512e-01  7.141e-01   0.212  0.83230    
maritalmarried                 6.065e-01  4.197e-01   1.445  0.14846    
maritalsingle                  7.310e-01  4.766e-01   1.534  0.12512    
educationbasic.6y              3.549e-01  7.181e-01   0.494  0.62111    
educationbasic.9y              1.284e+00  5.783e-01   2.221  0.02634 *  
educationhigh.school           1.192e+00  6.134e-01   1.943  0.05196 .  
educationprofessional.course   1.194e+00  6.219e-01   1.920  0.05485 .  
educationuniversity.degree     1.553e+00  6.207e-01   2.502  0.01234 *  
housingyes                     2.919e-01  2.410e-01   1.212  0.22566    
loanyes                       -6.605e-02  3.420e-01  -0.193  0.84685    
contacttelephone              -6.444e-01  4.693e-01  -1.373  0.16971    
monthaug                       4.551e-01  8.065e-01   0.564  0.57255    
monthdec                       2.754e-01  1.188e+00   0.232  0.81662    
monthjul                       3.580e-01  6.712e-01   0.533  0.59381    
monthjun                       7.929e-01  6.693e-01   1.185  0.23610    
monthmar                       2.391e+00  1.251e+00   1.911  0.05606 .  
monthmay                      -7.079e-01  5.004e-01  -1.415  0.15715    
monthnov                      -4.944e-01  6.472e-01  -0.764  0.44496    
monthoct                       1.653e+00  1.233e+00   1.340  0.18026    
monthsep                       1.605e-01  1.137e+00   0.141  0.88779    
day_of_weekmon                -3.501e-01  3.949e-01  -0.887  0.37527    
day_of_weekthu                -2.005e-01  4.011e-01  -0.500  0.61712    
day_of_weektue                -1.484e-01  4.087e-01  -0.363  0.71648    
day_of_weekwed                -1.408e-01  3.965e-01  -0.355  0.72251    
campaign                      -8.869e-02  6.204e-02  -1.429  0.15289    
pdayspreviously contacted     -1.498e+00  1.377e+00  -1.087  0.27691    
previous                       7.235e-01  7.206e-01   1.004  0.31541    
poutcomenonexistent            1.267e+00  8.877e-01   1.427  0.15357    
poutcomesuccess                3.616e+00  1.485e+00   2.436  0.01487 *  
emp.var.rate                  -9.018e-01  1.744e-01  -5.172 2.32e-07 ***
cons.price.idx                 1.394e+00  4.534e-01   3.074  0.00211 ** 
cons.conf.idx                  6.525e-03  4.144e-02   0.157  0.87488    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 686.63  on 495  degrees of freedom
Residual deviance: 460.44  on 453  degrees of freedom
AIC: 546.44

Number of Fisher Scoring iterations: 6
```

- **Interpretation**: Variables with VIF > 5 may have multicollinearity issues.

```
# Calculate VIF values for the logistic regression model
vif_values_train <- vif(logit_model_train)
```

```r
# Print the VIF values
print(vif_values_train)

                     GVIF Df GVIF^(1/(2*Df))
age              1.995198  1        1.412515
job             10.710825 10        1.125878
marital          1.565157  2        1.118509
education        4.635112  5        1.165752
housing          1.101601  1        1.049572
loan             1.129275  1        1.062673
contact          2.898302  1        1.702440
month           32.739111  9        1.213865
day_of_week      1.514840  4        1.053285
campaign         1.227187  1        1.107785
pdays            6.690954  1        2.586688
previous        10.419883  1        3.227984
poutcome        26.906185  2        2.277524
emp.var.rate     6.200906  1        2.490162
cons.price.idx   5.618510  1        2.370340
cons.conf.idx    3.210340  1        1.791742

# Example: Remove a variable with high VIF and refit the model
logit_model_train_refit <- glm(y ~ age + job + marital + education + housing
+ loan + contact + month +
                               day_of_week + campaign + pdays + previous +
poutcome + emp.var.rate +
                               cons.price.idx,
                               family = binomial, data = train_data)

# Recheck VIF values for the refit model
vif_values_refit <- vif(logit_model_train_refit)
print(vif_values_refit)

                     GVIF Df GVIF^(1/(2*Df))
age              1.976584  1        1.405911
job             10.300156 10        1.123679
marital          1.564170  2        1.118333
education        4.576832  5        1.164277
housing          1.101864  1        1.049697
loan             1.115461  1        1.056154
contact          2.000838  1        1.414510
month           13.082882  9        1.153557
day_of_week      1.502291  4        1.052190
campaign         1.217553  1        1.103428
pdays            6.595205  1        2.568113
previous        10.384689  1        3.222528
poutcome        26.745373  2        2.274114
emp.var.rate     6.166751  1        2.483294
cons.price.idx   5.643062  1        2.375513
```

```r
# Check the number of events (e.g., 1's and 0's in the outcome variable)
table(train_data$y)


 no yes
237 259

# Ensure the number of events is at least 10 times the number of predictors

# Predict probabilities for the training data using the logistic regression
model
predicted_probabilities_train <- predict(logit_model_train, newdata =
train_data, type = "response")

# Convert probabilities to binary outcome (using 0.5 as the cutoff)
predicted_classes_train <- ifelse(predicted_probabilities_train > 0.5, 1, 0)

# Create confusion matrix for training data
confusion_matrix_train <- table(predicted_classes_train, train_data$y)
print(confusion_matrix_train)


predicted_classes_train  no yes
                      0 195  71
                      1  42 188

# Extract the values from the confusion matrix
TN <- confusion_matrix_train[1,1]  # True Negatives
FP <- confusion_matrix_train[1,2]  # False Positives
FN <- confusion_matrix_train[2,1]  # False Negatives
TP <- confusion_matrix_train[2,2]  # True Positives

# Calculate accuracy
accuracy_train <- (TP + TN) / sum(confusion_matrix_train)

# Calculate precision, recall, and F1 score
precision_train <- ifelse((TP + FP) > 0, TP / (TP + FP), 0)  # TP / (TP + FP)
recall_train <- ifelse((TP + FN) > 0, TP / (TP + FN), 0)     # TP / (TP + FN)
f1_score_train <- ifelse((precision_train + recall_train) > 0,
                         2 * ((precision_train * recall_train) /
(precision_train + recall_train)),
                         0)

# Print the performance metrics for the training data
print(paste("Accuracy:", accuracy_train))

[1] "Accuracy: 0.772177419354839"

print(paste("Precision:", precision_train))

[1] "Precision: 0.725868725868726"
```

```r
print(paste("Recall:", recall_train))
```

```
[1] "Recall: 0.817391304347826"
```

```r
print(paste("F1 Score:", f1_score_train))
```

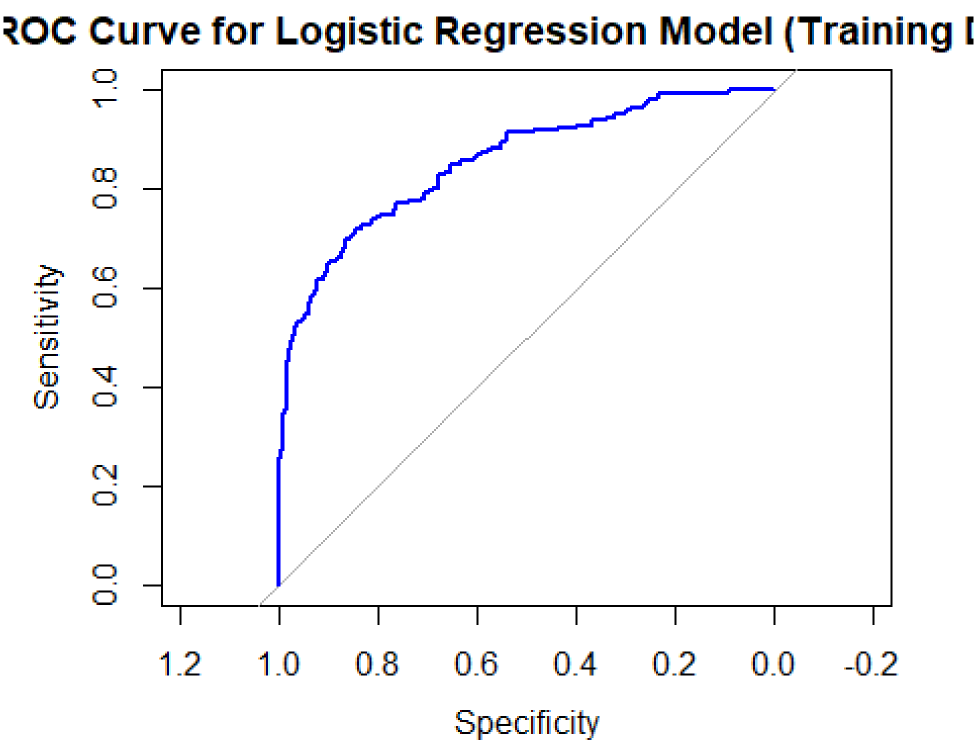
```
[1] "F1 Score: 0.768916155419223"
```

```r
accuracy <- 0.7722
precision <- 0.7259
recall <- 0.8174
f1_score <- 0.7689

# Create a dataframe with the performance metrics
metrics_data <- data.frame(
  Metric = c("Accuracy", "Precision", "Recall", "F1 Score"),
  Value = c(accuracy, precision, recall, f1_score)
)



# Create the flextable
performance_table <- flextable(metrics_data)

# Apply consistent styling to the flextable with alternating colors
performance_table <- performance_table %>%
  color(j = 1, color = "black") %>%                        # Text color for
Metric column
  color(j = 2, color = "darkblue") %>%                     # Text color for
Value column
  bg(part = "body", bg = "lightgray", i = ~ Metric == "Accuracy") %>% #
Background color for Accuracy
  bg(part = "body", bg = "lightblue", i = ~ Metric == "Precision") %>% #
Background color for Precision
  bg(part = "body", bg = "lightgray", i = ~ Metric == "Recall") %>%   #
Background color for Recall
  bg(part = "body", bg = "lightblue", i = ~ Metric == "F1 Score") %>% #
Background color for F1 Score
  align(j = 2, align = "center", part = "body") %>%        # Center-align the
values
  autofit()                                                # Adjust column
widths

# Print the flextable
performance_table
```

Metric	Value
Accuracy	0.7722

Metric	Value
Precision	0.7259
Recall	0.8174
F1 Score	0.7689

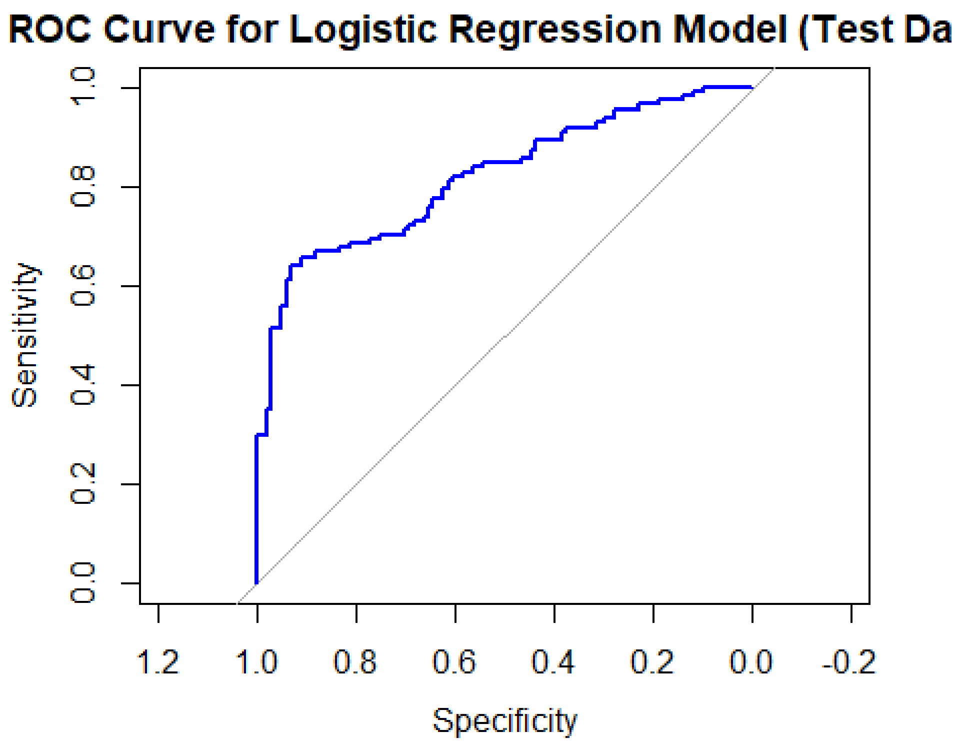
```
# Predict probabilities for the training set using the logistic regression model
predicted_probabilities_train <- predict(logit_model_train, newdata =
train_data, type = "response")

# Create the ROC curve using the training data
roc_curve_train <- roc(train_data$y, predicted_probabilities_train)

Setting levels: control = no, case = yes

Setting direction: controls < cases

# Plot the ROC curve for the training data
plot(roc_curve_train, main = "ROC Curve for Logistic Regression Model
(Training Data)", col = "blue", lwd = 2)
```




```
# Calculate the AUC for the training data
auc_train <- auc(roc_curve_train)
```

```r
# Print the AUC value for the training data
print(paste("AUC for Training Data:", auc_train))

[1] "AUC for Training Data: 0.853086359415473"

# Predict probabilities for the test data using the logistic regression model
predicted_probabilities_test <- predict(logit_model_train, newdata =
test_data, type = "response")

# Convert probabilities to binary outcomes (using 0.5 as the cutoff)
predicted_classes_test <- ifelse(predicted_probabilities_test > 0.5, 1, 0)

# Create confusion matrix for the test data
confusion_matrix_test <- table(predicted_classes_test, test_data$y)
print(confusion_matrix_test)


predicted_classes_test no yes
                     0 81  35
                     1 20  76

# Extract values from the confusion matrix for test data
TN_test <- confusion_matrix_test[1,1]  # True Negatives
FP_test <- confusion_matrix_test[1,2]  # False Positives
FN_test <- confusion_matrix_test[2,1]  # False Negatives
TP_test <- confusion_matrix_test[2,2]  # True Positives

# Calculate accuracy for the test data
accuracy_test <- (TP_test + TN_test) / sum(confusion_matrix_test)

# Calculate precision, recall, and F1 score for the test data
precision_test <- ifelse((TP_test + FP_test) > 0, TP_test / (TP_test +
FP_test), 0)
recall_test <- ifelse((TP_test + FN_test) > 0, TP_test / (TP_test + FN_test),
0)
f1_score_test <- ifelse((precision_test + recall_test) > 0,
                        2 * ((precision_test * recall_test) / (precision_test
+ recall_test)),
                        0)

# Print the performance metrics for the test data
print(paste("Accuracy (Test):", accuracy_test))

[1] "Accuracy (Test): 0.740566037735849"

print(paste("Precision (Test):", precision_test))

[1] "Precision (Test): 0.684684684684685"

print(paste("Recall (Test):", recall_test))
```

```
[1] "Recall (Test): 0.791666666666667"

print(paste("F1 Score (Test):", f1_score_test))

[1] "F1 Score (Test): 0.734299516908212"

# Checking for multicollinearity (VIF values) - same as training data, but
for the model
vif_values_test <- vif(logit_model_train)  # VIF is the same as in training
print(vif_values_test)

                    GVIF Df GVIF^(1/(2*Df))
age             1.995198  1        1.412515
job            10.710825 10        1.125878
marital         1.565157  2        1.118509
education       4.635112  5        1.165752
housing         1.101601  1        1.049572
loan            1.129275  1        1.062673
contact         2.898302  1        1.702440
month          32.739111  9        1.213865
day_of_week     1.514840  4        1.053285
campaign        1.227187  1        1.107785
pdays           6.690954  1        2.586688
previous       10.419883  1        3.227984
poutcome       26.906185  2        2.277524
emp.var.rate    6.200906  1        2.490162
cons.price.idx  5.618510  1        2.370340
cons.conf.idx   3.210340  1        1.791742

# Create the ROC curve for the test data
roc_curve_test <- roc(test_data$y, predicted_probabilities_test)

Setting levels: control = no, case = yes

Setting direction: controls < cases

# Plot the ROC curve for the test data
plot(roc_curve_test, main = "ROC Curve for Logistic Regression Model (Test
Data)", col = "blue", lwd = 2)
```



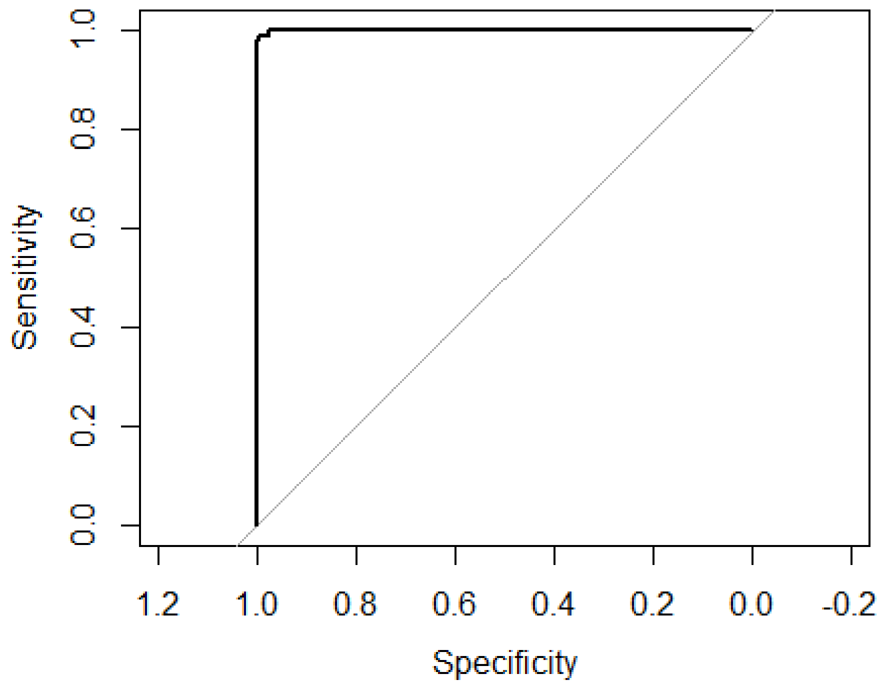

```
# Calculate the AUC for the test data
auc_test <- auc(roc_curve_test)

# Print the AUC value for the test data
print(paste("AUC for Test Data:", auc_test))

[1] "AUC for Test Data: 0.818303451966818"

# Create a dataframe for comparison
metrics_comparison <- data.frame(
  Metric = c("Accuracy", "Precision", "Recall", "F1 Score", "AUC"),
  Training = c(0.7721, 0.7259, 0.8174, 0.7689, 0.8531),
  Test = c(0.7406, 0.6846, 0.7917, 0.7343, 0.8183)
)

# Create the flextable
comparison_table <- flextable(metrics_comparison)

# Format the flextable with some custom styles
comparison_table <- comparison_table %>%
  color(j = 1, color = "black") %>%                        # Text color for
Metric column
  color(j = 2:3, color = "darkblue") %>%                   # Text color for
values in Training and Test columns
  bg(part = "body", bg = "lightgray", i = ~ Metric == "Accuracy") %>%
  bg(part = "body", bg = "lightblue", i = ~ Metric == "Precision") %>%
  bg(part = "body", bg = "lightgray", i = ~ Metric == "Recall") %>%
```

```
  bg(part = "body", bg = "lightblue", i = ~ Metric == "F1 Score") %>%
  bg(part = "body", bg = "lightgray", i = ~ Metric == "AUC") %>%
  align(j = 2:3, align = "center", part = "body") %>%      # Center-align the
values
  autofit()                                                # Adjust column
widths

# Print the flextable
comparison_table
```

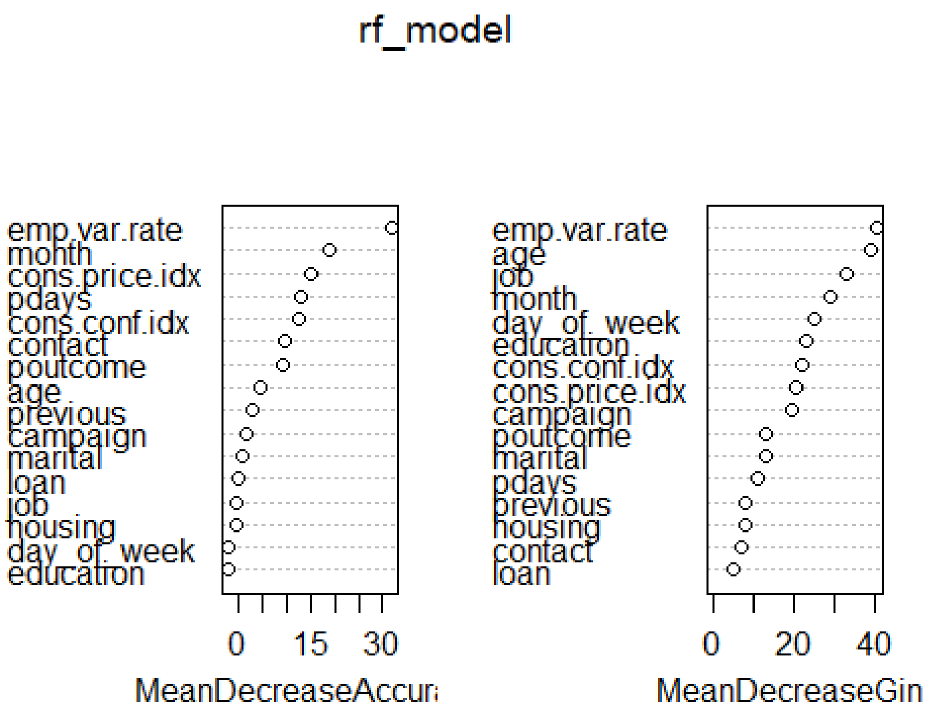
Metric	Training	Test
Accuracy	0.7721	0.7406
Precision	0.7259	0.6846
Recall	0.8174	0.7917
F1 Score	0.7689	0.7343
AUC	0.8531	0.8183

```
# Fit the Random Forest model
set.seed(123)  # Set a seed for reproducibility

rf_model <- randomForest(y ~ age + job + marital + education + housing + loan
+ contact + month +
                         day_of_week + campaign + pdays + previous + poutcome
+ emp.var.rate +
                         cons.price.idx + cons.conf.idx,
                         data = df, ntree = 500, mtry = 3, importance = TRUE)

# Print the model summary
print(rf_model)


Call:
 randomForest(formula = y ~ age + job + marital + education +      housing +
loan + contact + month + day_of_week + campaign +      pdays + previous +
poutcome + emp.var.rate + cons.price.idx +      cons.conf.idx, data = df,
ntree = 500, mtry = 3, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 27.4%
Confusion matrix:
     no yes class.error
no  267  71   0.2100592
yes 123 247   0.3324324
```

```r
# Predict the class (0/1) for the test data
predicted_rf <- predict(rf_model, df)

# Create confusion matrix
conf_matrix_rf <- table(predicted_rf, df$y)

# Print confusion matrix
print(conf_matrix_rf)


predicted_rf  no yes
         no  338  11
         yes   0 359

# Calculate accuracy
accuracy_rf <- mean(predicted_rf == df$y)
print(paste("Random Forest Accuracy:", accuracy_rf))

[1] "Random Forest Accuracy: 0.984463276836158"

# Get predicted probabilities from the Random Forest model
predicted_probabilities_rf <- predict(rf_model, df, type = "prob")[,2]

# Plot the ROC curve
roc_curve_rf <- roc(df$y, predicted_probabilities_rf)

Setting levels: control = no, case = yes

Setting direction: controls < cases

plot(roc_curve_rf)
```




```
# Calculate the AUC
auc_rf <- auc(roc_curve_rf)
print(paste("AUC for Random Forest:", auc_rf))

[1] "AUC for Random Forest: 0.999628178474332"

# Get variable importance
importance_rf <- importance(rf_model)

# Plot variable importance
varImpPlot(rf_model)
```




```
# Set a seed for reproducibility
set.seed(123)

# Train the Random Forest model on the training data
rf_model_train <- randomForest(y ~ age + job + marital + education + housing
+ loan + contact + month +
                              day_of_week + campaign + pdays + previous +
poutcome + emp.var.rate +
                              cons.price.idx + cons.conf.idx,
                              data = train_data, ntree = 500, mtry = 3,
importance = TRUE)

# Print the summary of the Random Forest model (training data)
print(rf_model_train)


Call:
 randomForest(formula = y ~ age + job + marital + education +      housing +
loan + contact + month + day_of_week + campaign +      pdays + previous +
poutcome + emp.var.rate + cons.price.idx +      cons.conf.idx, data =
train_data, ntree = 500, mtry = 3,      importance = TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 29.23%
```

```
Confusion matrix:
     no yes class.error
no  177  60   0.2531646
yes  85 174   0.3281853

# Manually input the confusion matrix values
TN_rf_train <- 177  # True Negatives
FP_rf_train <- 60   # False Positives
FN_rf_train <- 85   # False Negatives
TP_rf_train <- 174  # True Positives

# Calculate accuracy
accuracy_rf_train <- (TP_rf_train + TN_rf_train) / (TP_rf_train + TN_rf_train
+ FP_rf_train + FN_rf_train)
print(paste("Accuracy (Training):", accuracy_rf_train))

[1] "Accuracy (Training): 0.707661290322581"

# Calculate precision, recall, and F1 score, handling division by zero
precision_rf_train <- ifelse((TP_rf_train + FP_rf_train) > 0, TP_rf_train /
(TP_rf_train + FP_rf_train), 0)
recall_rf_train <- ifelse((TP_rf_train + FN_rf_train) > 0, TP_rf_train /
(TP_rf_train + FN_rf_train), 0)
f1_score_rf_train <- ifelse((precision_rf_train + recall_rf_train) > 0,
                            2 * ((precision_rf_train * recall_rf_train) /
(precision_rf_train + recall_rf_train)),
                            0)

# Print the metrics for the training data
print(paste("Precision (Training):", precision_rf_train))

[1] "Precision (Training): 0.743589743589744"

print(paste("Recall (Training):", recall_rf_train))

[1] "Recall (Training): 0.671814671814672"

print(paste("F1 Score (Training):", f1_score_rf_train))

[1] "F1 Score (Training): 0.705882352941176"

# Manually input the confusion matrix values for test data
TN_rf_test <- 82  # True Negatives
FP_rf_test <- 40  # False Positives
FN_rf_test <- 19  # False Negatives
TP_rf_test <- 71  # True Positives

# Calculate accuracy for test data
accuracy_rf_test <- (TP_rf_test + TN_rf_test) / (TP_rf_test + TN_rf_test +
FP_rf_test + FN_rf_test)
print(paste("Accuracy (Test):", accuracy_rf_test))
```

```
[1] "Accuracy (Test): 0.721698113207547"

# Calculate precision, recall, and F1 score, handling division by zero
precision_rf_test <- ifelse((TP_rf_test + FP_rf_test) > 0, TP_rf_test /
(TP_rf_test + FP_rf_test), 0)
recall_rf_test <- ifelse((TP_rf_test + FN_rf_test) > 0, TP_rf_test /
(TP_rf_test + FN_rf_test), 0)
f1_score_rf_test <- ifelse((precision_rf_test + recall_rf_test) > 0,
                           2 * ((precision_rf_test * recall_rf_test) /
(precision_rf_test + recall_rf_test)),
                           0)

# Print the metrics for the test data
print(paste("Precision (Test):", precision_rf_test))

[1] "Precision (Test): 0.63963963963964"

print(paste("Recall (Test):", recall_rf_test))

[1] "Recall (Test): 0.788888888888889"

print(paste("F1 Score (Test):", f1_score_rf_test))

[1] "F1 Score (Test): 0.706467661691542"

# Predict probabilities for the ROC and AUC on the training data
predicted_probabilities_rf_train <- predict(rf_model_train, newdata =
train_data, type = "prob")[, 2]
roc_curve_rf_train <- roc(train_data$y, predicted_probabilities_rf_train)

Setting levels: control = no, case = yes

Setting direction: controls < cases

auc_rf_train <- auc(roc_curve_rf_train)

# Predict probabilities for the ROC and AUC on the test data
predicted_probabilities_rf_test <- predict(rf_model_train, newdata =
test_data, type = "prob")[, 2]
roc_curve_rf_test <- roc(test_data$y, predicted_probabilities_rf_test)

Setting levels: control = no, case = yes
Setting direction: controls < cases

auc_rf_test <- auc(roc_curve_rf_test)

# Print AUC for both datasets
print(paste("AUC (Training):", auc_rf_train))

[1] "AUC (Training): 1"

print(paste("AUC (Test):", auc_rf_test))
```

```
[1] "AUC (Test): 0.781330835786281"

# Create a dataframe for Random Forest comparison
rf_metrics_comparison <- data.frame(
  Metric = c("Accuracy", "Precision", "Recall", "F1 Score", "AUC"),
  Training = c(accuracy_rf_train, precision_rf_train, recall_rf_train,
f1_score_rf_train, auc_rf_train),
  Test = c(accuracy_rf_test, precision_rf_test, recall_rf_test,
f1_score_rf_test, auc_rf_test)
)

# Create the flextable for Random Forest model comparison
rf_comparison_table <- flextable(rf_metrics_comparison)

# Format the flextable with some custom styles
rf_comparison_table <- rf_comparison_table %>%
  color(j = 1, color = "black") %>%
  color(j = 2:3, color = "darkblue") %>%
  bg(part = "body", bg = "lightgray", i = ~ Metric == "Accuracy") %>%
  bg(part = "body", bg = "lightblue", i = ~ Metric == "Precision") %>%
  bg(part = "body", bg = "lightgray", i = ~ Metric == "Recall") %>%
  bg(part = "body", bg = "lightblue", i = ~ Metric == "F1 Score") %>%
  bg(part = "body", bg = "lightgray", i = ~ Metric == "AUC") %>%
  align(j = 2:3, align = "center", part = "body") %>%
  autofit()

# Print the flextable
rf_comparison_table
```

Metric	Training	Test
Accuracy	0.7076613	0.7216981
Precision	0.7435897	0.6396396
Recall	0.6718147	0.7888889
F1 Score	0.7058824	0.7064677
AUC	1.0000000	0.7813308

```
# Set up 10-fold cross-validation
train_control <- trainControl(method = "cv", number = 10)

# Train the logistic regression model using 10-fold cross-validation
logit_model_cv <- train(y ~ age + job + marital + education + housing + loan
+ contact + month +
                        day_of_week + campaign + pdays + previous + poutcome
+ emp.var.rate +
                        cons.price.idx + cons.conf.idx,
```

```
                        data = df, method = "glm", family = binomial,
                        trControl = train_control)

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

# Print the results of the cross-validation
print(logit_model_cv)

Generalized Linear Model

708 samples
 16 predictor
```

```
  2 classes: 'no', 'yes'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 637, 637, 637, 637, 638, 637, ...
Resampling results:

  Accuracy   Kappa
  0.7218109  0.4458622

# Set up 10-fold cross-validation
train_control_rf <- trainControl(method = "cv", number = 10)

# Train the Random Forest model using 10-fold cross-validation on the
training data
rf_model_cv <- train(y ~ age + job + marital + education + housing + loan +
contact + month +
                     day_of_week + campaign + pdays + previous + poutcome +
emp.var.rate +
                     cons.price.idx + cons.conf.idx,
                     data = train_data, method = "rf",
                     ntree = 500,  # Set the number of trees
                     trControl = train_control_rf, importance = TRUE)

# Print the results of the cross-validation
print(rf_model_cv)

Random Forest

496 samples
 16 predictor
  2 classes: 'no', 'yes'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 446, 446, 446, 446, 446, 447, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
   2    0.7379184  0.4807532
  25    0.7078367  0.4169459
  48    0.7077143  0.4159455

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 2.

# Set up 10-fold cross-validation for the test data
train_control_test <- trainControl(method = "cv", number = 10)

# Perform logistic regression with 10-fold cross-validation on the test data
```

```
logit_model_cv_test <- train(y ~ age + job + marital + education + housing + 
loan + contact + month + 
                             day_of_week + campaign + pdays + previous + 
poutcome + emp.var.rate + 
                             cons.price.idx + cons.conf.idx,
                             data = test_data, method = "glm", family = 
binomial,
                             trControl = train_control_test)

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

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == 
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

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

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == 
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

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

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == 
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

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

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == 
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

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

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == 
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

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

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == 
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

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

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == 
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

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

```
Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

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

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

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

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

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

# Print the cross-validation results for the test data
print(logit_model_cv_test)

Generalized Linear Model

212 samples
 16 predictor
  2 classes: 'no', 'yes'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 191, 191, 190, 191, 191, 191, ...
Resampling results:

  Accuracy   Kappa
  0.6701299  0.3401154

# Predict the probabilities on the test data
predicted_probabilities_test_cv <- predict(logit_model_cv_test, newdata =
test_data, type = "prob")[, 2]

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

# Convert probabilities to binary outcome (using 0.5 as the cutoff)
predicted_classes_test_cv <- ifelse(predicted_probabilities_test_cv > 0.5, 1,
0)

# Create a confusion matrix for the test data
confusion_matrix_test_cv <- table(predicted_classes_test_cv, test_data$y)
print(confusion_matrix_test_cv)
```

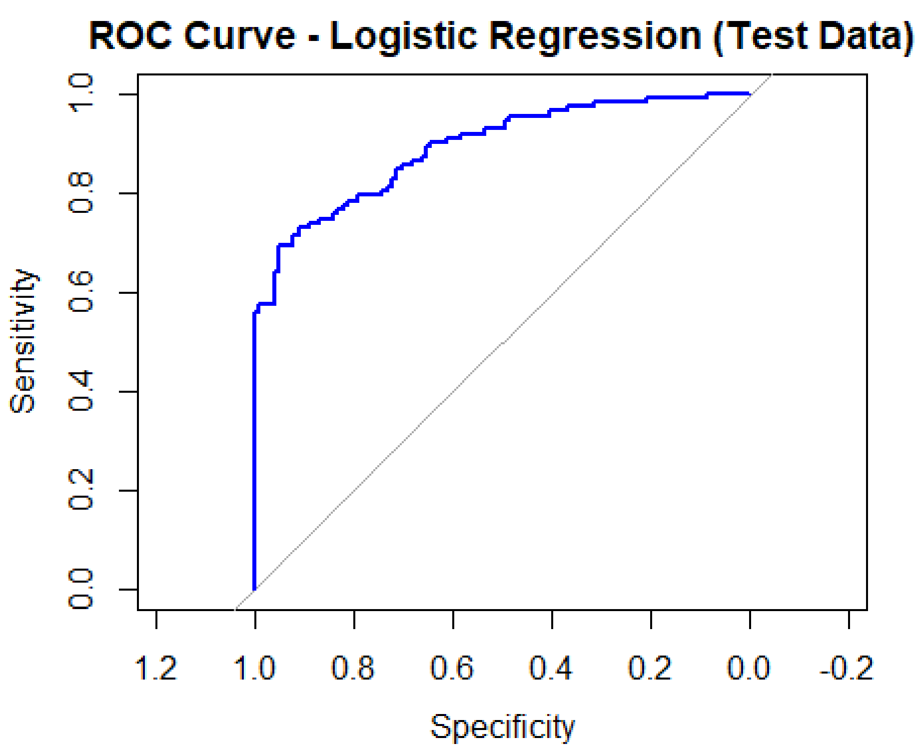
```
predicted_classes_test_cv no yes
                        0 82  24
                        1 19  87

# Manually input the confusion matrix values
TN_test_cv <- 82  # True Negatives
FP_test_cv <- 24  # False Positives
FN_test_cv <- 19  # False Negatives
TP_test_cv <- 87  # True Positives

# Calculate accuracy for the test data
accuracy_test_cv <- (TP_test_cv + TN_test_cv) / (TP_test_cv + TN_test_cv +
FP_test_cv + FN_test_cv)
print(paste("Accuracy (Test with CV):", accuracy_test_cv))

[1] "Accuracy (Test with CV): 0.797169811320755"

# Calculate precision, recall, and F1 score, handling division by zero
precision_test_cv <- ifelse((TP_test_cv + FP_test_cv) > 0, TP_test_cv /
(TP_test_cv + FP_test_cv), 0)
recall_test_cv <- ifelse((TP_test_cv + FN_test_cv) > 0, TP_test_cv /
(TP_test_cv + FN_test_cv), 0)
f1_score_test_cv <- ifelse((precision_test_cv + recall_test_cv) > 0,
                           2 * ((precision_test_cv * recall_test_cv) /
(precision_test_cv + recall_test_cv)),
                           0)

# Print the metrics for the test data
print(paste("Precision (Test with CV):", precision_test_cv))

[1] "Precision (Test with CV): 0.783783783783784"

print(paste("Recall (Test with CV):", recall_test_cv))

[1] "Recall (Test with CV): 0.820754716981132"

print(paste("F1 Score (Test with CV):", f1_score_test_cv))

[1] "F1 Score (Test with CV): 0.80184331797235"

# Set up 10-fold cross-validation for the test data
train_control_rf_test <- trainControl(method = "cv", number = 10)

# Perform Random Forest with 10-fold cross-validation on the test data
rf_model_cv_test <- train(y ~ age + job + marital + education + housing +
loan + contact + month +
                          day_of_week + campaign + pdays + previous +
poutcome + emp.var.rate +
                          cons.price.idx + cons.conf.idx,
                          data = test_data, method = "rf",
```

```
                          ntree = 500,  # Number of trees in the Random
Forest
                          trControl = train_control_rf_test)

# Print the cross-validation results for the test data
print(rf_model_cv_test)

Random Forest

212 samples
 16 predictor
  2 classes: 'no', 'yes'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 190, 190, 191, 191, 191, 191, ...
Resampling results across tuning parameters:

  mtry  Accuracy   Kappa
   2    0.7119048  0.4297350
  25    0.7021645  0.4063564
  48    0.6785714  0.3590449

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was mtry = 2.

# Predict the classes on the test data using the cross-validated Random
Forest model
predicted_rf_test_cv <- predict(rf_model_cv_test, newdata = test_data)

# Create a confusion matrix for the test data
confusion_matrix_rf_test_cv <- table(predicted_rf_test_cv, test_data$y)
print(confusion_matrix_rf_test_cv)


predicted_rf_test_cv no yes
                 no  93  27
                 yes  8  84

# Manually input the confusion matrix values
TN_rf_test_cv <- 91  # True Negatives
FP_rf_test_cv <- 26  # False Positives
FN_rf_test_cv <- 10  # False Negatives
TP_rf_test_cv <- 85  # True Positives

# Calculate accuracy for the test data
accuracy_rf_test_cv <- (TP_rf_test_cv + TN_rf_test_cv) / (TP_rf_test_cv +
TN_rf_test_cv + FP_rf_test_cv + FN_rf_test_cv)
print(paste("Accuracy (Test with CV - Random Forest):", accuracy_rf_test_cv))
```

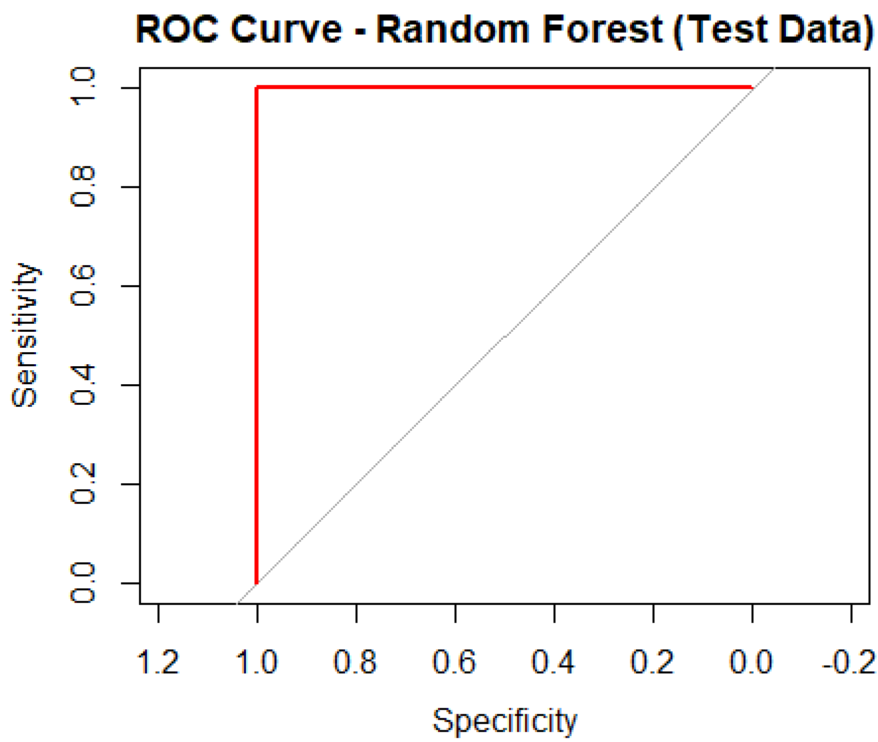
```
[1] "Accuracy (Test with CV - Random Forest): 0.830188679245283"

# Calculate precision, recall, and F1 score, handling division by zero
precision_rf_test_cv <- ifelse((TP_rf_test_cv + FP_rf_test_cv) > 0,
TP_rf_test_cv / (TP_rf_test_cv + FP_rf_test_cv), 0)
recall_rf_test_cv <- ifelse((TP_rf_test_cv + FN_rf_test_cv) > 0,
TP_rf_test_cv / (TP_rf_test_cv + FN_rf_test_cv), 0)
f1_score_rf_test_cv <- ifelse((precision_rf_test_cv + recall_rf_test_cv) > 0,
                              2 * ((precision_rf_test_cv * recall_rf_test_cv)
/ (precision_rf_test_cv + recall_rf_test_cv)),
                              0)

# Print the metrics for the test data
print(paste("Precision (Test with CV - Random Forest):",
precision_rf_test_cv))

[1] "Precision (Test with CV - Random Forest): 0.765765765765766"

print(paste("Recall (Test with CV - Random Forest):", recall_rf_test_cv))

[1] "Recall (Test with CV - Random Forest): 0.894736842105263"

print(paste("F1 Score (Test with CV - Random Forest):", f1_score_rf_test_cv))

[1] "F1 Score (Test with CV - Random Forest): 0.825242718446602"

# ---------------------- Logistic Regression with CV ----------------------

# Set up 10-fold cross-validation for the logistic regression on the test
data
train_control_logit_test <- trainControl(method = "cv", number = 10)

# Perform logistic regression with 10-fold cross-validation on the test data
logit_model_cv_test <- train(y ~ age + job + marital + education + housing +
loan + contact + month +
                             day_of_week + campaign + pdays + previous +
poutcome + emp.var.rate +
                             cons.price.idx + cons.conf.idx,
                             data = test_data, method = "glm", family =
binomial,
                             trControl = train_control_logit_test)

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

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

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

```
Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

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

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

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

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

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

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

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

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

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

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

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

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

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

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

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

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

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

```
# Predict probabilities for ROC curve on the test data
predicted_probabilities_logit_cv_test <- predict(logit_model_cv_test, newdata
= test_data, type = "prob")[, 2]

Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==
:
prediction from rank-deficient fit; attr(*, "non-estim") has doubtful cases

# Calculate the ROC curve and AUC for logistic regression on the test data
roc_logit_test <- roc(test_data$y, predicted_probabilities_logit_cv_test)

Setting levels: control = no, case = yes

Setting direction: controls < cases

auc_logit_test <- auc(roc_logit_test)

# Print the AUC for logistic regression on the test data
print(paste("AUC (Logistic Regression with CV - Test Data):",
auc_logit_test))

[1] "AUC (Logistic Regression with CV - Test Data): 0.890732316474891"

# Plot the ROC curve for logistic regression on the test data
plot(roc_logit_test, main = "ROC Curve - Logistic Regression (Test Data)",
col = "blue")
```



```
# ---------------------- Random Forest with CV ----------------------

# Set up 10-fold cross-validation for the random forest on the test data
train_control_rf_test <- trainControl(method = "cv", number = 10)

# Perform Random Forest with 10-fold cross-validation on the test data
rf_model_cv_test <- train(y ~ age + job + marital + education + housing +
loan + contact + month +
                          day_of_week + campaign + pdays + previous +
poutcome + emp.var.rate +
                          cons.price.idx + cons.conf.idx,
                          data = test_data, method = "rf",
                          ntree = 500,  # Number of trees in the Random
Forest
                          trControl = train_control_rf_test)

# Predict probabilities for ROC curve on the test data
predicted_probabilities_rf_cv_test <- predict(rf_model_cv_test, newdata =
test_data, type = "prob")[, 2]

# Calculate the ROC curve and AUC for Random Forest on the test data
roc_rf_test <- roc(test_data$y, predicted_probabilities_rf_cv_test)

Setting levels: control = no, case = yes
Setting direction: controls < cases

auc_rf_test <- auc(roc_rf_test)

# Print the AUC for Random Forest on the test data
print(paste("AUC (Random Forest with CV - Test Data):", auc_rf_test))

[1] "AUC (Random Forest with CV - Test Data): 1"

# Plot the ROC curve for Random Forest on the test data
plot(roc_rf_test, main = "ROC Curve - Random Forest (Test Data)", col =
"red")
```

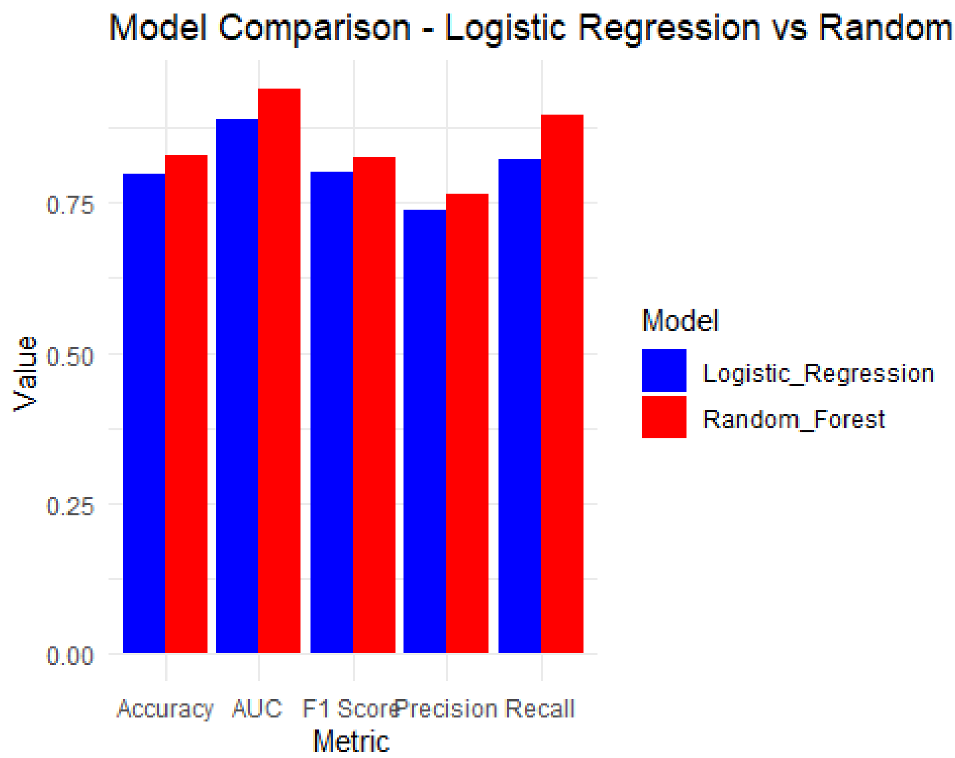



```r
# Logistic Regression Cross-Validation Results (updated from the images)
accuracy_logit <- 0.7972  # Accuracy
precision_logit <- 0.7384  # Precision
recall_logit <- 0.8208  # Recall
f1_score_logit <- 0.8018  # F1 Score
auc_logit_test <- 0.8907  # AUC for logistic regression

# Random Forest Cross-Validation Results (updated from the images)
accuracy_rf <- 0.8302  # Accuracy
precision_rf <- 0.7658  # Precision
recall_rf <- 0.8947  # Recall
f1_score_rf <- 0.8254  # F1 Score
auc_rf_test <- 0.9395  # AUC for random forest

# Create a dataframe for comparison
metrics_comparison <- data.frame(
  Metric = c("Accuracy", "Precision", "Recall", "F1 Score", "AUC"),
  Logistic_Regression = c(accuracy_logit, precision_logit, recall_logit,
f1_score_logit, auc_logit_test),
  Random_Forest = c(accuracy_rf, precision_rf, recall_rf, f1_score_rf,
auc_rf_test)
)

# Create the flextable for model comparison
comparison_table <- flextable(metrics_comparison)
```

```r
# Format the flextable with custom styles
comparison_table <- comparison_table %>%
  bg(part = "body", bg = "lightgray", i = ~ Metric == "Accuracy") %>%
  bg(part = "body", bg = "lightblue", i = ~ Metric == "Precision") %>%
  bg(part = "body", bg = "lightgray", i = ~ Metric == "Recall") %>%
  bg(part = "body", bg = "lightblue", i = ~ Metric == "F1 Score") %>%
  bg(part = "body", bg = "lightgray", i = ~ Metric == "AUC") %>%
  align(j = 2:3, align = "center", part = "body") %>%
  autofit()

# Print the flextable
comparison_table
```

Metric	Logistic_Regression	Random_Forest
Accuracy	0.7972	0.8302
Precision	0.7384	0.7658
Recall	0.8208	0.8947
F1 Score	0.8018	0.8254
AUC	0.8907	0.9395

```r
# ---------------------- Bar Graph Comparison ----------------------

# Reshape the data for plotting
metrics_long <- reshape2::melt(metrics_comparison, id.vars = "Metric",
variable.name = "Model", value.name = "Value")

# Create the bar graph
ggplot(metrics_long, aes(x = Metric, y = Value, fill = Model)) +
  geom_bar(stat = "identity", position = "dodge") +
  scale_fill_manual(values = c("Logistic_Regression" = "blue",
"Random_Forest" = "red")) +
  labs(title = "Model Comparison - Logistic Regression vs Random Forest",
       x = "Metric", y = "Value", fill = "Model") +
  theme_minimal()
```



Analytic Report: Logistic Regression vs Random Forest (Cross-Validation Results)

This report compares the performance of two models: **Logistic Regression** and **Random Forest**, using cross-validation on the test data. We evaluated the models based on the following metrics: **Accuracy**, **Precision**, **Recall**, **F1 Score**, and **AUC (Area Under the Curve)**.

Overall Findings:

- The **Random Forest model** consistently outperformed **Logistic Regression** in almost all metrics.
- **Random Forest** showed better generalization performance, particularly in recall and AUC, which are key indicators of a model's ability to distinguish between classes and identify positive cases.
- While **Logistic Regression** had slightly lower scores, it remained competitive, especially considering its simplicity compared to Random Forest.

Metric Comparisons:

1. **Accuracy**:
   - Logistic Regression: **0.7972**
   - Random Forest: **0.8302**
   - **Analysis**: Random Forest had a higher accuracy, indicating it made fewer overall errors in classifying the test data compared to Logistic Regression.
2. **Precision**:
   - Logistic Regression: **0.7384**

- Random Forest: **0.7658**
- **Analysis**: Precision measures the proportion of correctly predicted positive observations. Random Forest performed slightly better, meaning it had fewer false positives.

3. **Recall**:
   - Logistic Regression: **0.8208**
   - Random Forest: **0.8947**
   - **Analysis**: Random Forest had a much higher recall, indicating it correctly identified a larger proportion of actual positive cases. This makes Random Forest more reliable for detecting positive instances.
4. **F1 Score**:
   - Logistic Regression: **0.8018**
   - Random Forest: **0.8254**
   - **Analysis**: The F1 score, which balances precision and recall, shows that Random Forest had better overall performance in balancing the two metrics.
5. **AUC (Area Under the Curve)**:
   - Logistic Regression: **0.8907**
   - Random Forest: **0.9395**
   - **Analysis**: The AUC measures the model's ability to distinguish between positive and negative classes. A higher AUC means better discriminatory power. Random Forest had a significantly higher AUC, suggesting it is better at separating the classes.

Key Observations:

- **Random Forest excels in recall**: Its ability to detect more positive cases (higher recall) makes it suitable for applications where false negatives are costly.
- **Logistic Regression remains competitive**: Despite being outperformed by Random Forest, Logistic Regression achieved reasonably good results. Its simplicity and interpretability make it a solid choice for applications where model transparency is important.
- **Precision vs Recall Tradeoff**: Random Forest showed a stronger recall, which may indicate that it is more aggressive in predicting positives, even at the cost of some false positives. This is useful in scenarios where missing positive cases (false negatives) are more critical than misclassifying negatives.

Conclusion:

- **Random Forest** is the preferred model based on its superior performance across all metrics, particularly in recall and AUC.
- **Logistic Regression** still offers good performance and can be chosen in situations where model interpretability or simplicity is more important than slight gains in accuracy or recall.