STATS/CSE 780

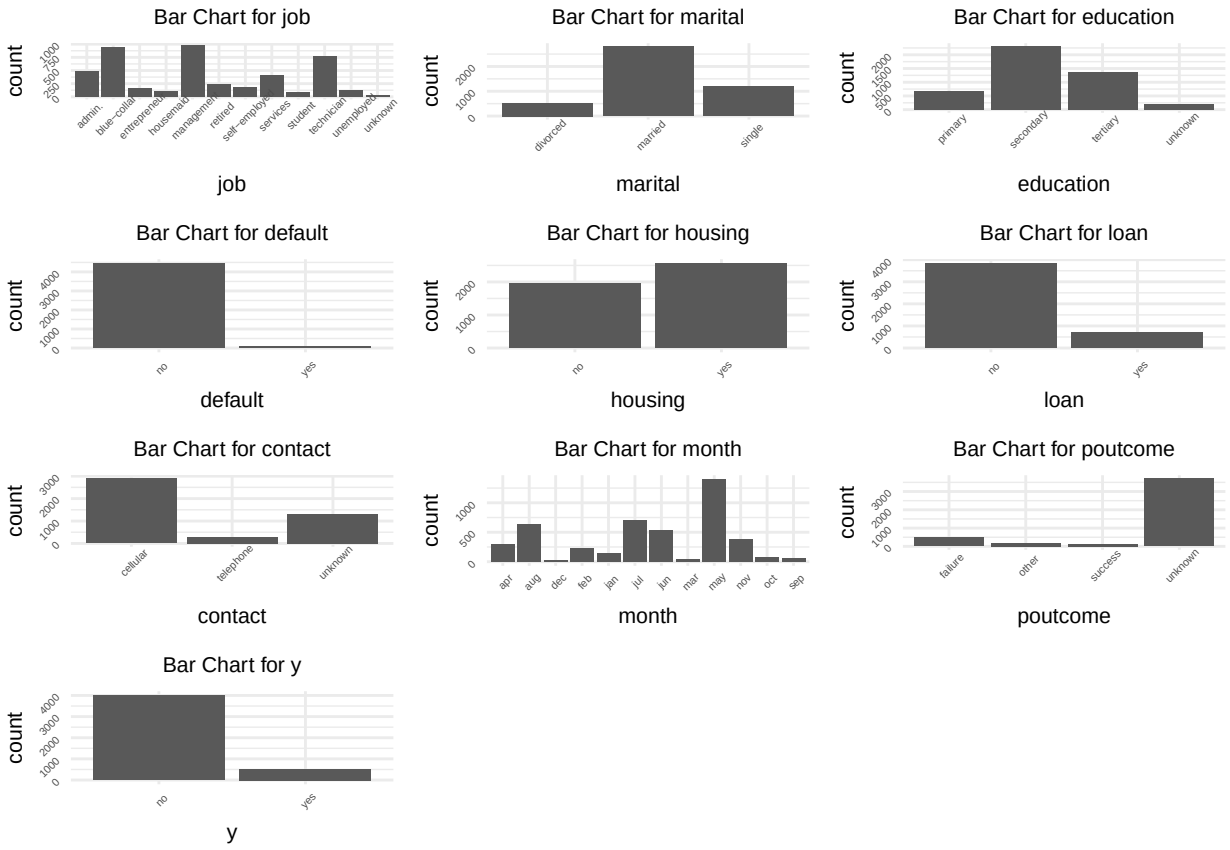
Supplementary of Project 1

Seyed Mohammad Mehdi Hassani Najafabadi(Student ID: 400489126)

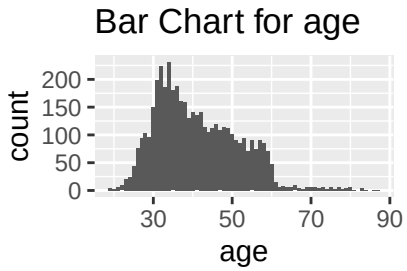
17 April, 2023

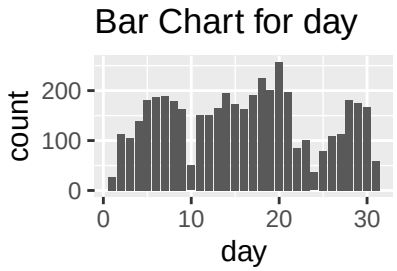
```r
# Load Bank Marketing Archive dataset
original_data<- read_delim("bank.csv",delim = ";",show_col_types = FALSE)
```

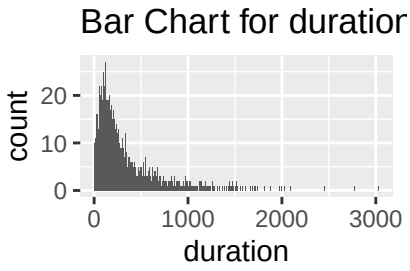
```r
cat_variables <- c('job', 'marital', 'education', 'default', 'housing',
                   'loan', 'contact', 'month','poutcome', 'y')
plots <- list()
for (var in cat_variables) {
  plot <- ggplot(clean_data, aes_string(x = var)) + geom_bar() +
    ggtitle(paste("Bar Chart for", var))+theme(plot.title = element_text(hjust = 0.5)) +
  theme_minimal()+
    theme(plot.title = element_text(hjust = 0.5, size = 8),
            axis.title = element_text(size = 8),
            axis.text = element_text(size = 4,angle=45))
  plots[[var]] <- plot
}
grid.arrange(grobs = plots, ncol = 3)
```

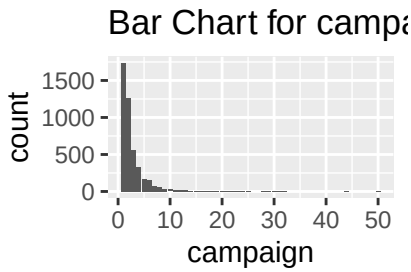


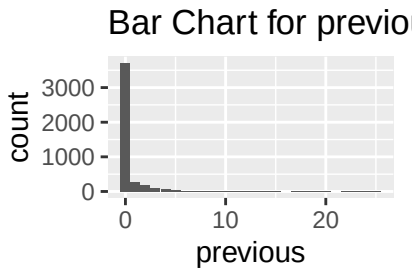

```
Num_variables <- c('age', 'day','duration', 'campaign', 'previous')
continuous_vars <- c('balance', 'pdays')
plots <- list()
for (var in Num_variables) {
  plot <- ggplot(clean_data, aes_string(x = var)) + geom_bar() +
    ggtitle(paste("Bar Chart for", var))
  plots[[var]] <- plot
}
# Loop through the continuous variables and create histograms
for (var in continuous_vars) {
  plot <- ggplot(clean_data, aes_string(x = var)) + geom_histogram(bins = 30) + ggtitle(paste('
  plots[[var]] <- plot
}
grid.arrange(grobs = plots, ncol = 3)
```

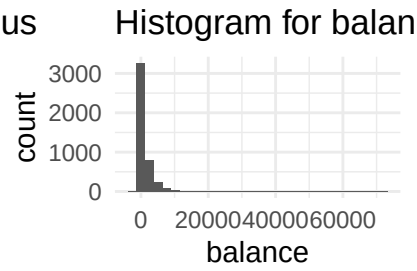

Bar Chart for age
count
200
150
100
50
0
30
50
70
90
age

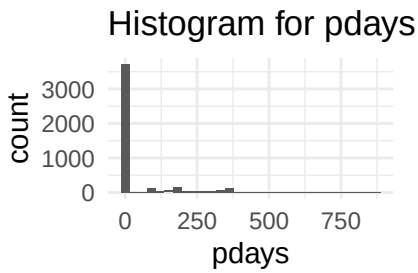

Bar Chart for day
count
200
100
0
0
10
20
30
day

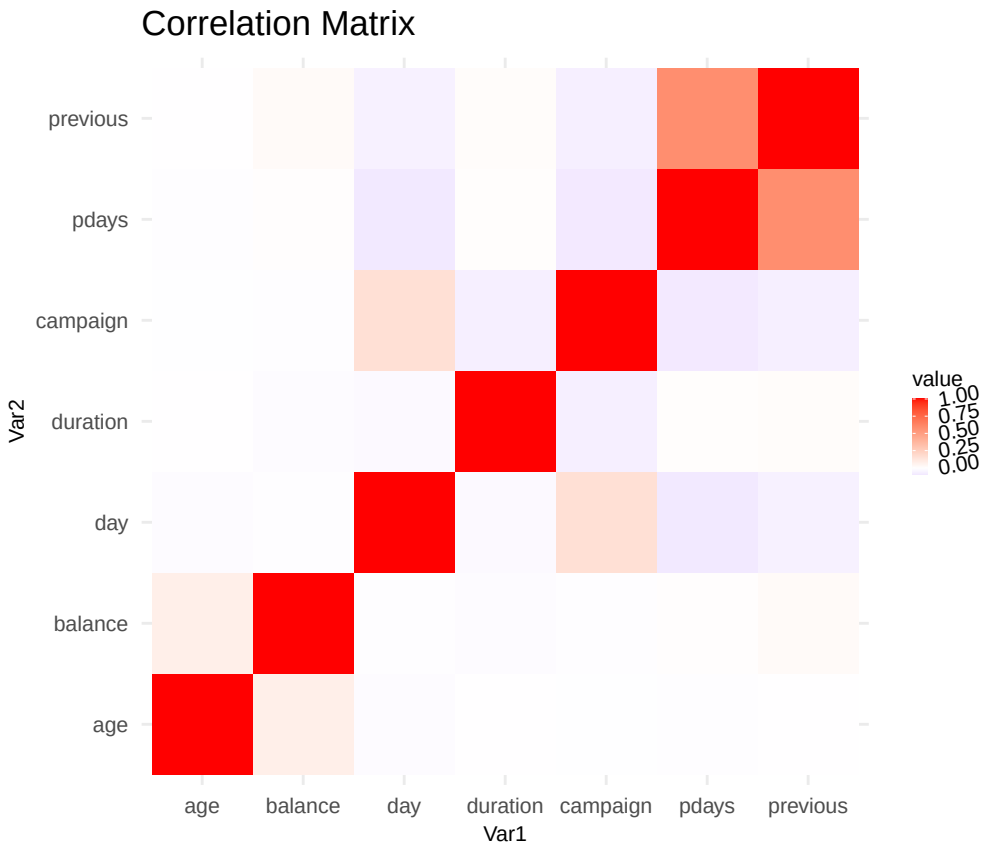

Bar Chart for duratior
count
20
10
0
0
1000
2000
3000
duration


Bar Chart for campa
count
1500
1000
500
0
0
10
20
30
40
50
campaign


Bar Chart for previous
count
3000
2000
1000
0
0
10
20
previous


Histogram for balan
count
3000
2000
1000
0
0
200004000060000
balance

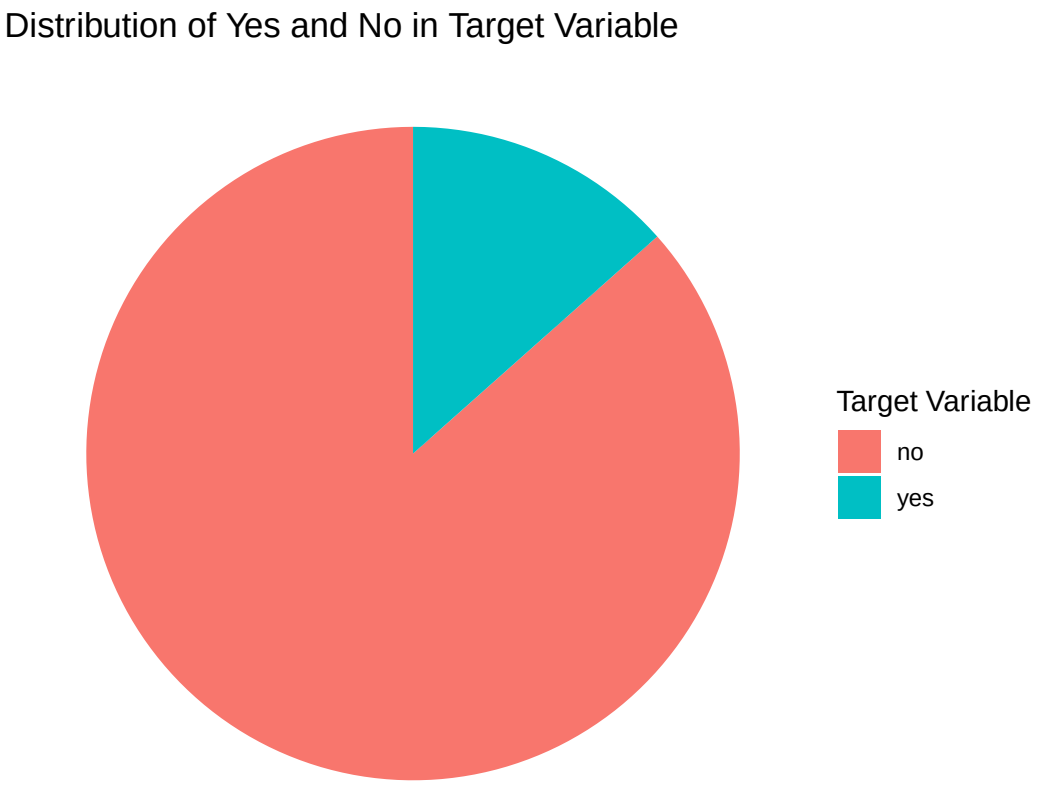

Histogram for pdays
count
3000
2000
1000
0
0
250
500
750
pdays



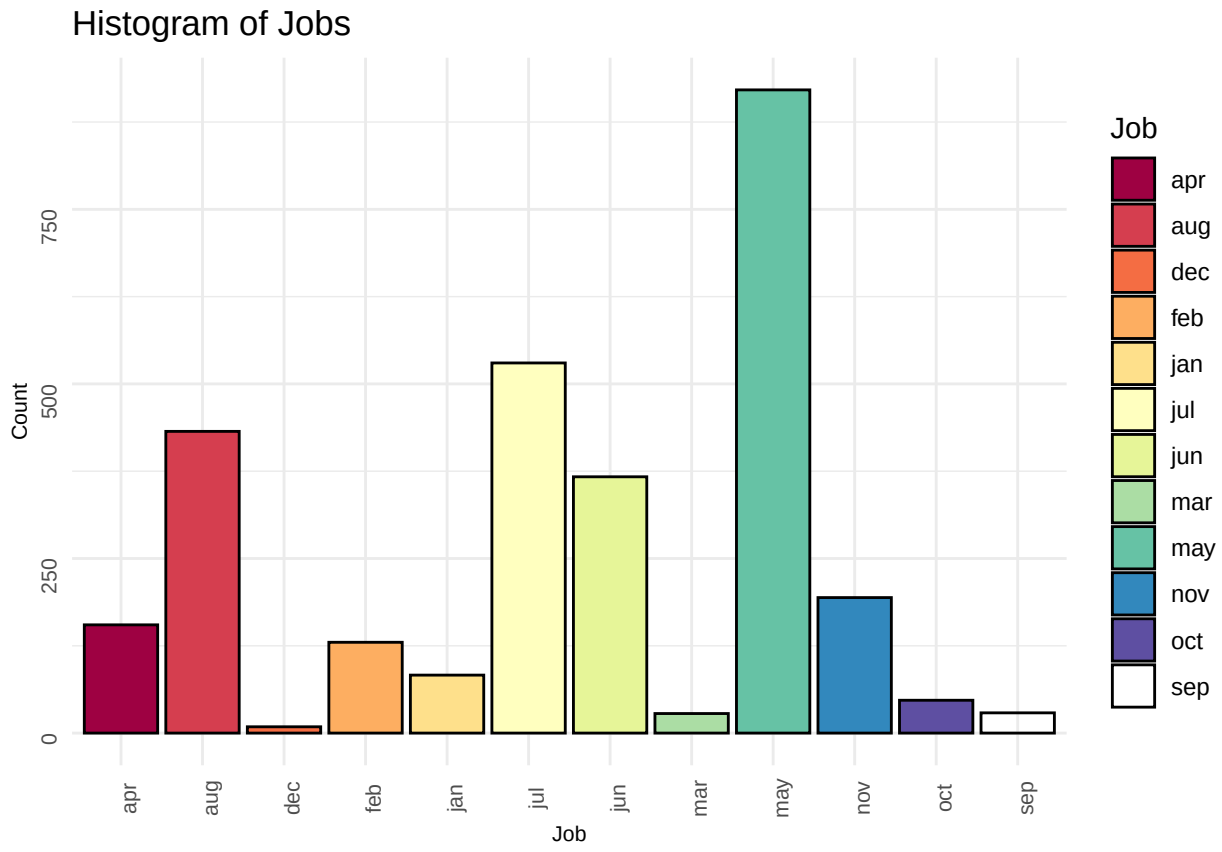

```r
# Create a function to filter outliers based on IQR
filter_outliers <- function(data) {
  numeric_columns <- sapply(data, is.numeric)
  numeric_data <- data[, numeric_columns]
  bounds <- t(sapply(numeric_data, function(x) {
    IQR_x <- IQR(x, na.rm = TRUE)
    Q1 <- quantile(x, 0.25, na.rm = TRUE)
    Q3 <- quantile(x, 0.75, na.rm = TRUE)
    lower_bound <- Q1 - 1.5 * IQR_x
    upper_bound <- Q3 + 1.5 * IQR_x
    return(c(lower_bound, upper_bound))
  }))
  within_bounds <- mapply(function(column, lower, upper) {
    column >= lower & column <= upper
  }, numeric_data, bounds[, 1], bounds[, 2])
```

```r
  # Consider only rows that have no outliers in numeric columns
  no_outliers_indices <- rowSums(within_bounds) == ncol(numeric_data)
  # Filter the dataset based on the calculated indices
  no_outliers_data <- data[no_outliers_indices, ]
  return(no_outliers_data)
}
# Split data based on target variable
data_yes <- clean_data[clean_data$y == "yes", ]
data_no <- clean_data[clean_data$y == "no", ]
# Apply the filter_outliers function to each class
filtered_data_yes <- filter_outliers(data_yes)
filtered_data_no <- filter_outliers(data_no)
# Merge the filtered datasets
filtered_data <- rbind(filtered_data_yes, filtered_data_no)
```

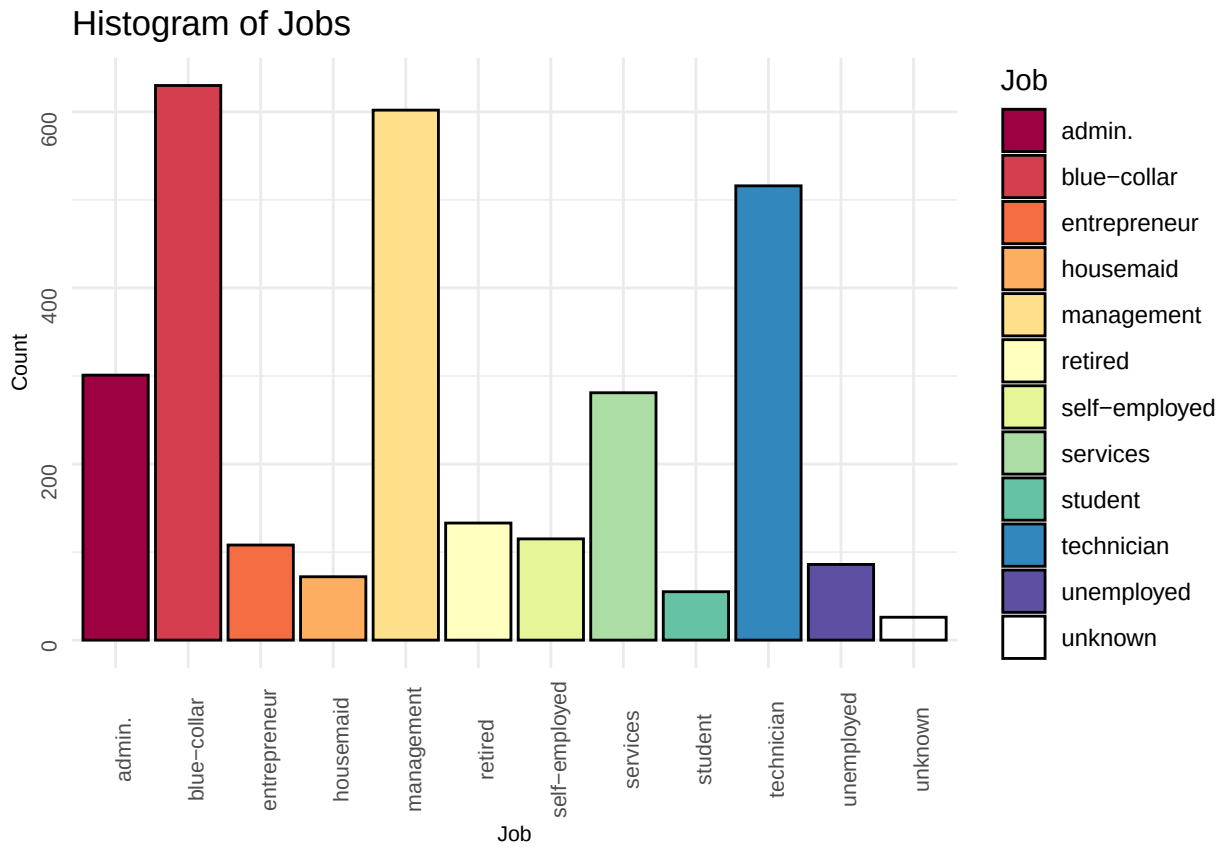
```r
# Convert the target variable to a factor and count the occurrences of each level
target_counts <- as.data.frame(table(filtered_data$y))
# Create a pie chart
ggplot(target_counts, aes(x = "", y = Freq, fill = Var1)) +
  geom_bar(stat = "identity", width = 0.01) +
  coord_polar("y", start = 0) +theme_minimal()+
  theme_void() +
  theme(legend.position = "right") +
  labs(fill = "Target Variable", title = "Distribution of Yes and No in Target Variable")
```



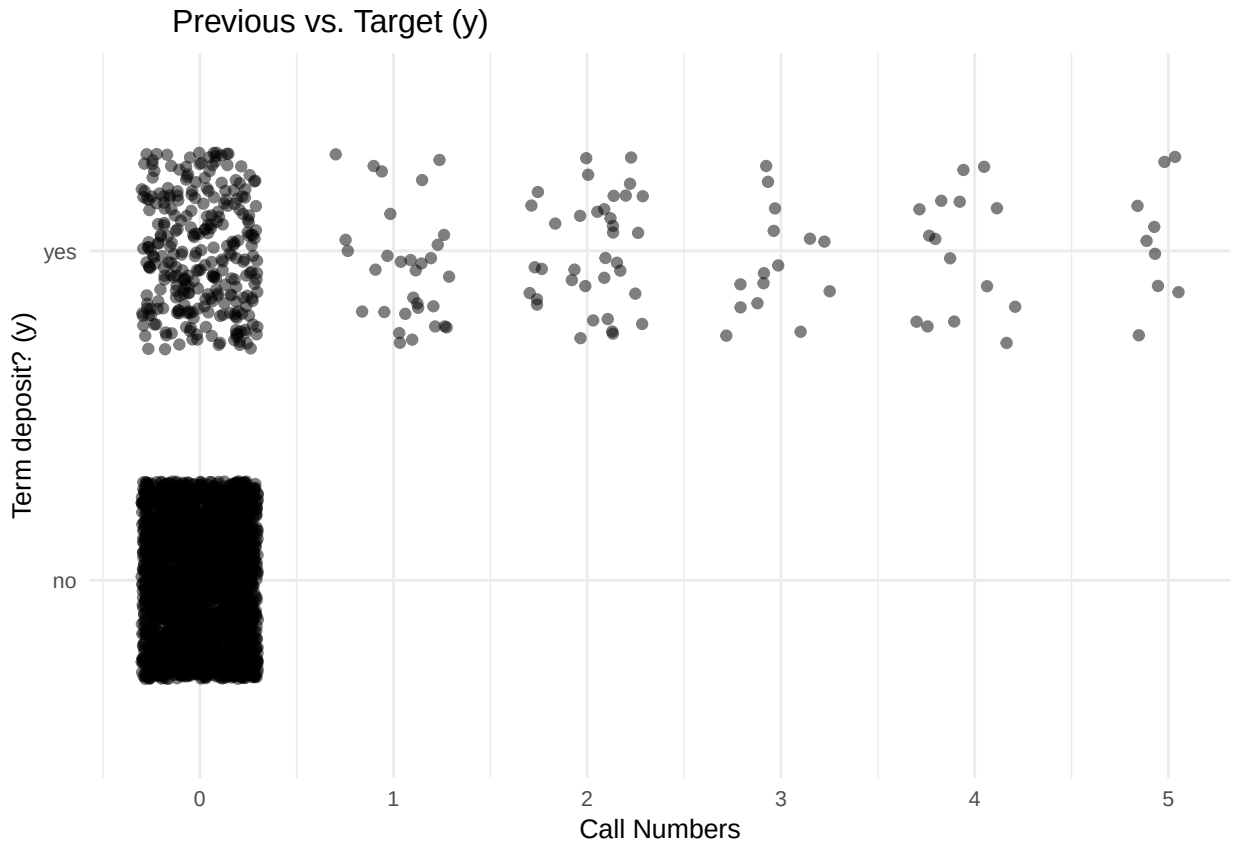

```
# Create a colorful histogram for the 'month' variable
ggplot(filtered_data, aes(x = filtered_data$month, fill = filtered_data$month)) +
  geom_histogram(stat = "count", position = "dodge", binwidth = 0.25, color = "black") +
  theme(plot.title = element_text(hjust = 0.1)) +
   theme(plot.margin = unit(c(0, 0, 0, 0), "cm") ) +
  theme_minimal() +theme(
            axis.title = element_text(size = 8),
            axis.text = element_text(size = 8,angle=90))+
  labs(x = "Job", y = "Count", title = "Histogram of Jobs", fill = "Job") +
  scale_fill_brewer(palette = "Spectral")
```




```
# Create a colorful histogram for the 'job' variable
ggplot(filtered_data, aes(x = filtered_data$job, fill = filtered_data$job)) +
  geom_histogram(stat="count",position="dodge",binwidth =1,color ="black")+     theme(plot.titl
   theme(plot.margin = unit(c(0, 0, 0, 0), "cm") ) +
  theme_minimal() +theme(
            axis.title = element_text(size = 8),
            axis.text = element_text(size = 8,angle=90))+
  labs(x = "Job", y = "Count", title = "Histogram of Jobs", fill = "Job") +
  scale_fill_brewer(palette = "Spectral")
```




```
# Create a plot of the 'previous' variable versus the target 'y' variable
ggplot(filtered_data, aes(x = previous, y = y)) +
  geom_jitter(alpha = 0.5, width = 0.3, height = 0.3) +
  theme_minimal() +
  labs(x = "Call Numbers", y = "Term deposit? (y)", title = "Previous vs. Target (y)") +
  theme(
    plot.title = element_text(hjust = 0.1, size = 12),
    axis.title = element_text(size = 10),
    axis.text = element_text(size = 8)
    )
```




```r
# Define your data, target variable (y), and the number of folds for cross-validation
y <- filtered_data$y
k <- 5
# Create folds for cross-validation
set.seed(123)
folds <- createFolds(y, k = k, list = TRUE, returnTrain = FALSE)
# Initialize variables to store performance metrics for the Decision Trees classifier
dt_acc <- numeric(k)
# Find the best train and test sets based on Decision Trees performance
best_acc <- 0
best_train_set <- NULL
best_test_set <- NULL
for (i in 1:k) {
  # Create train and test sets for the current fold
  train_set <- filtered_data[-folds[[i]], ]
```

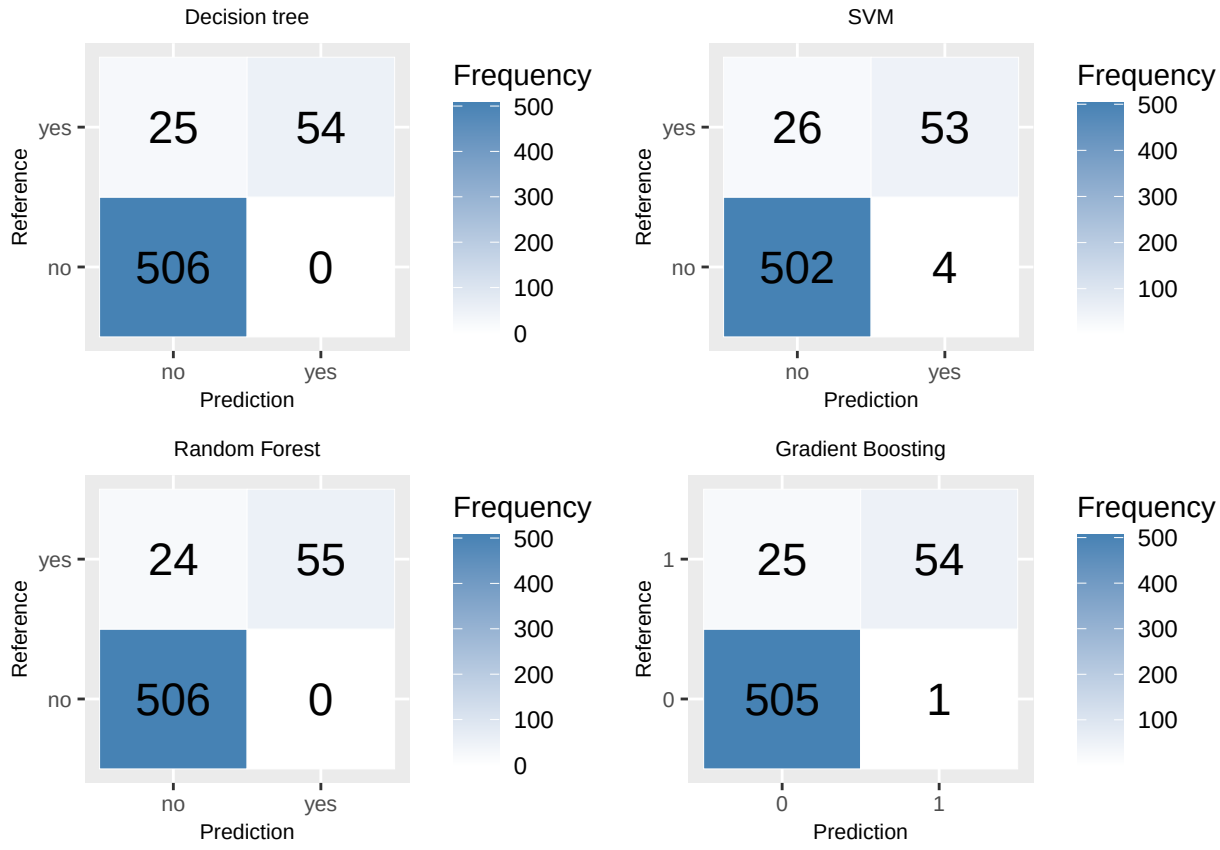
```
  test_set <- filtered_data[folds[[i]], ]
    # Decision Trees
  dt_model <- rpart(y ~ ., data = train_set, method = "class")
  dt_pred <- predict(dt_model, test_set, type = "class")
  dt_acc[i] <- sum(dt_pred == test_set$y) / nrow(test_set)
    # Check if the current fold has the highest accuracy
  if (dt_acc[i] > best_acc) {
    best_acc <- dt_acc[i]
    best_train_set <- train_set
    best_test_set <- test_set
  }
}
```

```
confusion_matrix_plot <- function(matrices, titles) {
  n <- length(matrices)
  plots_list <- list()
  for (i in 1:n) {
    matrix <- matrices[[i]]
    title <- titles[[i]]

    matrix_df <- as.data.frame(matrix)
    matrix_df$Freq_Label <- paste0(matrix_df$Freq)
    p <- ggplot(matrix_df, aes(x = Prediction, y = Reference)) +
      geom_tile(aes(fill = Freq), color = "white") +
      scale_fill_gradient(low = "white", high = "steelblue") +
      geom_text(aes(label = Freq_Label), size = 6, color = "black") +
      labs(title = title, x = "Prediction", y = "Reference", fill = "Frequency") +
      theme(plot.title = element_text(hjust = 0.5, size = 8),
            axis.title = element_text(size = 8),
            axis.text = element_text(size = 8))

    plots_list[[i]] <- p
```

```
  }


  return(plots_list)
}
# Decision Trees
dt_model <- rpart(y ~ ., data = best_train_set, method = "class")
dt_pred <- predict(dt_model, best_test_set, type = "class")
dt_acc <- sum(dt_pred == best_test_set$y) / nrow(best_test_set)
dt_precision <- precision(as.factor(best_test_set$y), as.factor(dt_pred))
dt_recall <- recall(as.factor(best_test_set$y), as.factor(dt_pred))
dt_f1 <- 2 * dt_precision * dt_recall / (dt_precision + dt_recall)
# Confusion Matrix
dt_cm <- confusionMatrix(dt_pred, as.factor(best_test_set$y))$table
#=============================================================================
# SVM
svm_model <- svm(as.factor(y) ~ ., data = best_train_set,
                 kernel = "linear", probability = TRUE)
svm_pred <- predict(svm_model, best_test_set, type = "class")
svm_acc <- sum(svm_pred == best_test_set$y) / nrow(best_test_set)
svm_precision <- precision(as.factor(best_test_set$y), as.factor(svm_pred))
svm_recall <- recall(as.factor(best_test_set$y), as.factor(svm_pred))
svm_f1 <- 2 * svm_precision * svm_recall / (svm_precision + svm_recall)
# Confusion Matrix
svm_cm <- confusionMatrix(svm_pred, as.factor(best_test_set$y))$table
#=============================================================================
# Random Forest
best_train_set$y <- as.factor(best_train_set$y)
best_test_set$y <- as.factor(best_test_set$y)
rf_model <- randomForest(y ~ ., data = best_train_set, ntree = 100, method = "class")
rf_pred <- predict(rf_model, best_test_set)
rf_acc <- sum(rf_pred == best_test_set$y) / nrow(best_test_set)
```

```r
rf_precision <- precision(as.factor(best_test_set$y), as.factor(rf_pred))
rf_recall <- recall(as.factor(best_test_set$y), as.factor(rf_pred))
rf_f1 <- 2 * rf_precision * rf_recall / (rf_precision + rf_recall)
# Confusion Matrix
rf_cm <- confusionMatrix(rf_pred, as.factor(best_test_set$y))$table
#==========================================================================
# Gradient Boosting Classifier
best_train_set$y <- as.factor(best_train_set$y)
best_train_set$y <- as.numeric(best_train_set$y) - 1
best_test_set$y <- as.numeric(best_test_set$y) - 1
train_matrix <- xgb.DMatrix(data.matrix(best_train_set[
  , -which(colnames(best_train_set) == "y")])
  , label = best_train_set$y)
test_matrix <- xgb.DMatrix(data.matrix(best_test_set[
  ,-which(colnames(best_test_set) == "y")])
  , label = best_test_set$y)
# Set the parameters for XGBoost
xgb_params <- list(
  objective = "binary:logistic",
  eval_metric = "logloss",
  eta = 0.1,
  max_depth = 6,
  min_child_weight = 1,
  subsample = 0.8,
  colsample_bytree = 0.8
)
# Train the XGBoost model
  xgb_model <- xgb.train(
    params = xgb_params,
    data = train_matrix,
    nrounds = 100
```

```
  )
    # Make predictions on the test set
  xgb_pred <- predict(xgb_model, test_matrix)
  xgb_pred <- ifelse(xgb_pred > 0.5, 1, 0)
  xgb_acc <- sum(xgb_pred == best_test_set$y) / nrow(best_test_set)
  xgb_precision <- precision(as.factor(best_test_set$y), as.factor(xgb_pred))
  xgb_recall <- recall(as.factor(best_test_set$y), as.factor(xgb_pred))
  xgb_f1 <- 2 * xgb_precision * xgb_recall / (xgb_precision + xgb_recall)
  # Confusion Matrix
xgb_cm <- confusionMatrix(as.factor(xgb_pred), as.factor(best_test_set$y))$table
 #=============================================================================
  # Plot Confusion Matrix
  Confusion_Matrix <- list(dt_cm, svm_cm, rf_cm, xgb_cm)
  titles <- list("Decision tree", "SVM", "Random Forest", "Gradient Boosting")
# Create the list of ggplot objects
plots_list <- confusion_matrix_plot(Confusion_Matrix, titles)
# Determine the layout based on the number of plots
num_plots <- length(plots_list)
num_rows <- ceiling(sqrt(num_plots))
num_cols <- ceiling(num_plots / num_rows)
# Arrange the plots in a grid layout
grid.arrange(grobs = plots_list, nrow = num_rows, ncol = num_cols)
```




```
# Create a data frame to store the evaluation metrics
metrics <- data.frame(
  classifier = c("Decision Trees", "SVM", "Random Forest", "XGBoost"),
  accuracy = c(dt_acc, svm_acc, rf_acc, xgb_acc),
  f1_score = c(dt_f1, svm_f1, rf_f1, xgb_f1),
  precision = c(dt_precision, svm_precision, rf_precision, xgb_precision),
  recall = c(dt_recall, svm_recall, rf_recall, xgb_recall)
)
# Calculate the average rank of each classifier across all metrics
metrics[, c("accuracy_rank", "f1_score_rank", "precision_rank", "recall_rank")] <-
  apply(metrics[, c("accuracy", "f1_score", "precision", "recall")], 2, rank)
metrics$Rank <- rowMeans(metrics[,
                                 c("accuracy_rank", "f1_score_rank", "precision_rank", "recall_
# Sort the classifiers based on the average rank
metrics <- metrics[order(metrics$Rank,decreasing=FALSE), ]
```

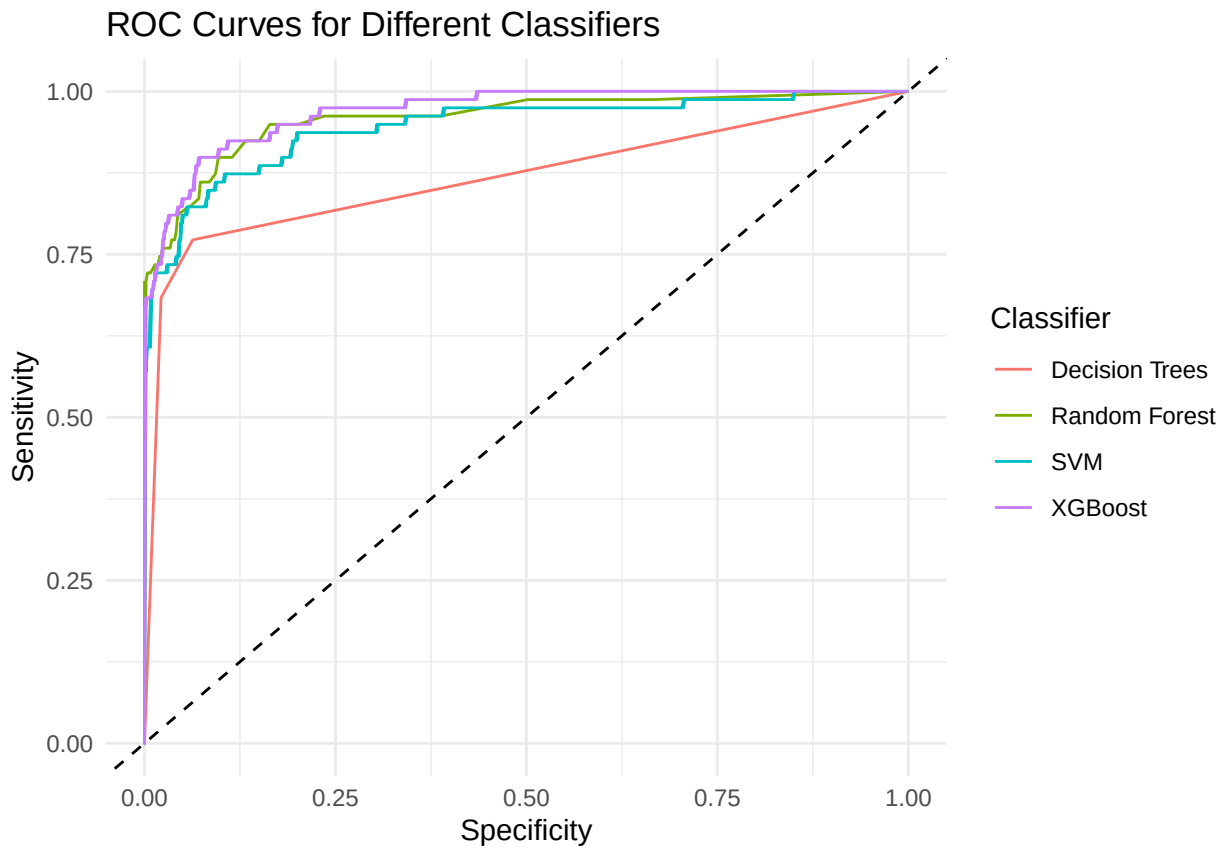
```r
# Create a well-organized table for the report
# Remove the third column
metrics <- metrics[, c(-6,-7,-8,-9,-10)]
colnames(metrics)[0] <- "Rank"
# Set the font size, row height, and column width
font_size <- "\\small"
row_height <- "\\renewcommand{\\arraystretch}{1.5}"
column_widths <- c("0.5cm", "3cm", "2cm","2cm","2cm","2cm","0cm","0cm","0cm","0cm","2cm")
formatted_table <- metrics %>%
  kable("latex", caption = "Classifier Evaluation Metrics and Ranking", align = "c") %>%
  kable_styling(latex_options = c("striped", "hold_position")) %>%
  row_spec(0, bold = TRUE, color = "white", background = "gray") %>%
  column_spec(1, bold = TRUE, width = column_widths[1]) %>%
  column_spec(2, width = column_widths[2]) %>%
  column_spec(3, width = column_widths[3])%>%
  column_spec(4, width = column_widths[4]) %>%
  column_spec(5, width = column_widths[5])%>%
  column_spec(6, width = column_widths[6])%>%
   column_spec(7, width = column_widths[7])%>%
   column_spec(8, width = column_widths[8])%>%
   column_spec(9, width = column_widths[9])%>%
   column_spec(10, width = column_widths[10])%>%
  column_spec(11, width = column_widths[11])
# Print the table
formatted_table
```

Table 1: Classifier Evaluation Metrics and Ranking

	classifier	accuracy	f1_score	precision	recall
2	SVM	0.9487179	0.9709865	0.9920949	0.9507576
4	XGBoost	0.9555556	0.9749035	0.9980237	0.9528302
1	Decision Trees	0.9572650	0.9758920	1.0000000	0.9529190
3	Random Forest	0.9589744	0.9768340	1.0000000	0.9547170

```r
 # Assuming the classifiers are trained and the predictions are available
# Replace the placeholders with the actual prediction probabilities for each classifier
dt_prob <- predict(dt_model, best_test_set, type = "prob")[, 2];
svm_model <- svm(as.factor(y) ~ ., data = best_train_set,
                 kernel = "linear", probability = TRUE);
svm_prob <-
  as.vector(attr(predict(svm_model, best_test_set, probability = TRUE), "probabilities")[, 2])
rf_prob <- predict(rf_model, best_test_set, type = "prob")[, 2];
xgb_prob <- predict(xgb_model, test_matrix);

# Calculate the True Positive Rate (TPR) and False Positive Rate (FPR)
dt_roc <- roc(best_test_set$y, dt_prob);
svm_roc <- roc(best_test_set$y, svm_prob);
rf_roc <- roc(best_test_set$y, rf_prob);
xgb_roc <- roc(best_test_set$y, xgb_prob);

# Create a dataframe to store the ROC data
roc_data <- data.frame(
  TPR = c(dt_roc$sensitivities,
          svm_roc$sensitivities, rf_roc$sensitivities, xgb_roc$sensitivities),
  FPR = c(dt_roc$specificities,
          svm_roc$specificities, rf_roc$specificities, xgb_roc$specificities),
  Classifier = c(rep("Decision Trees", length(dt_roc$sensitivities)),
                 rep("SVM", length(svm_roc$sensitivities)),
                 rep("Random Forest", length(rf_roc$sensitivities)),
                 rep("XGBoost", length(xgb_roc$sensitivities)))
)
# Plot the ROC curves
ggplot(roc_data, aes(x = 1 - FPR, y = TPR, color = Classifier)) +
  geom_line() +
  geom_abline(intercept = 0, slope = 1, linetype = "dashed") +
  labs(title = "ROC Curves for Different Classifiers",
```

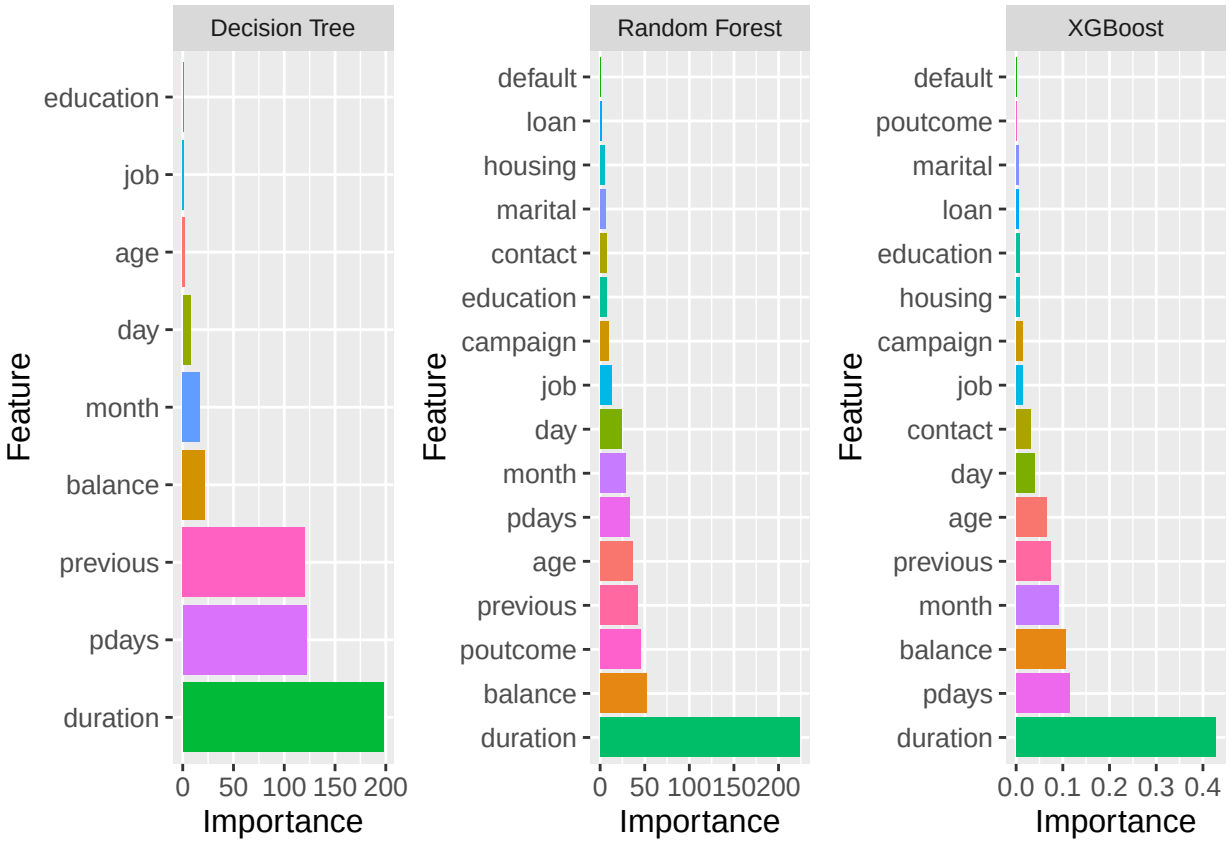
```
       x = "Specificity", y = "Sensitivity", color = "Classifier") +
  theme_minimal()
```




```
# Get feature importance from the Decision Tree model
dt_importance <- as.data.frame(dt_model$variable.importance)
colnames(dt_importance) <- "importance"
dt_importance$feature <- row.names(dt_importance)
dt_importance <- dt_importance[order(-dt_importance$importance), ]
# Get feature importance from the Random Forest model
rf_importance <- as.data.frame(importance(rf_model))
colnames(rf_importance) <- "importance"
rf_importance$feature <- row.names(rf_importance)
rf_importance <- rf_importance[order(-rf_importance$importance), ]
# Get feature importance from the XGBoost model
xgb_importance <-
```

```
  xgb.importance(feature_names = colnames(best_train_set[, -which(colnames(best_train_set) == "
xgb_importance$feature <- row.names(xgb_importance)

plot_feature_importance <- function(importance_data, model_name) {
  ggplot(importance_data, aes(x = reorder(feature, -importance), y = importance)) +
    geom_bar(stat = "identity", fill = "steelblue") +
    coord_flip() +
    labs(title = paste0("Feature Importance for ", model_name),
         x = "Feature",
         y = "Importance") +
    theme(plot.title = element_text(hjust = 0.5, size = 14),
          axis.title = element_text(size = 12),
          axis.text = element_text(size = 10))
}

# Ensure the same structure for all data frames
dt_importance$Model <- "Decision Tree"
rf_importance$Model <- "Random Forest"
xgb_importance$Model <- "XGBoost"
xgb_importance <- xgb_importance[,-c(3,4)]
colnames(xgb_importance)[c(1,2)] <- c("feature", "importance")
dt_importance <- dt_importance[, c("feature", "importance", "Model")]
rf_importance <- rf_importance[, c("feature", "importance", "Model")]
xgb_importance <- xgb_importance[,c("feature", "importance", "Model")]
# Combine feature importance data frames
#all_importance <- rbind(dt_importance, rf_importance, xgb_importance)
# Adjust the plot creation function
plot_feature_importance <- function(importance_data) {
  ggplot(importance_data, aes(x = reorder(feature, -importance),
                              y = importance, fill = feature)) +
    geom_bar(stat = "identity", position = "dodge") +
```

```
    coord_flip() +
    labs(
         x = "Feature",
         y = "Importance") +
    theme(plot.title = element_text(hjust = 0.5, size = 14),
          axis.title = element_text(size = 12),
          axis.text = element_text(size = 10)) +
    facet_wrap(~Model, scales = "free_y", ncol = 3) +
    guides(fill = FALSE)
}
# Create the combined feature importance plot
dt_importance_plot <- plot_feature_importance(dt_importance)
rf_importance_plot <- plot_feature_importance(rf_importance)
xgb_importance_plot <- plot_feature_importance(xgb_importance)
#ggtitle(dt_importance_plot, "Decision Tree Feature Importance")
#ggtitle(rf_importance_plot, "Random Forest Feature Importance")
#ggtitle(xgb_importance_plot, "XGBoost Feature Importance")
# Display the plots in one row
grid.arrange(dt_importance_plot, rf_importance_plot, xgb_importance_plot, ncol = 3)
```


Decision Tree
education
job
age
day
month
balance
previous
pdays
duration
Feature
0 50 100 150 200
Importance
Random Forest
default
loan
housing
marital
contact
education
campaign
job
day
month
pdays
age
previous
poutcome
balance
duration
Feature
0 50 100150200
Importance
XGBoost
default
poutcome
marital
loan
education
housing
campaign
job
contact
day
age
previous
month
balance
pdays
duration
Feature
0.0 0.1 0.2 0.3 0.4
Importance