Milestone 3: Random Forest Tree

Nosipho Precious Donkrag, Nontsikelelo Sharon Buhlungu, Tshepang Mogosi, Pitsi Pitsi

2024-10-18

Random Forest Tree

```
library(readr)
library(ggplot2)
library(gridExtra)
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:gridExtra':
##
##     combine
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggcorrplot)
library(fastDummies)
library(corrplot)
```

```
## corrplot 0.94 loaded
```

```
setwd("C:/Users/nosip/Documents/third Year/BIN381/milestones")
```

```
data_df <- read_csv("data_for_ml.csv", show_col_types = FALSE)
names(data_df)
```

```
##  [1] "marital_status"     "household_size"     "yrs_of_residence"
##  [4] "Annual_Salary"      "Months_Annual"      "FRS.Contribution"
##  [7] "Net_Salary"         "Net_months"         "Gross_Salary"
## [10] "Gross_Months"       "Qualify"            "Education_Bach."
## [13] "Education_HS-grad"  "Education_Masters"  "Occupation_Cleric."
## [16] "Occupation_Exec."   "Occupation_Prof."   "Occupation_Sales"
## [19] "age"
```

```
#rename column for naming convention
colnames(data_df)[13] <- "Education_HS_grad"
```

The following columns will be converted to factors; witth the aim of determining how they will affect the ml algorithm:

```
columns_to_convert <- c("marital_status",  "Qualify",
                        "Education_Bach.", "Education_HS_grad", "Education_Masters",
                        "Occupation_Cleric.", "Occupation_Exec.", "Occupation_Prof.", "Occupation_Sales
columns_to_convert
```

```
## [1] "marital_status"     "Qualify"            "Education_Bach."
## [4] "Education_HS_grad"  "Education_Masters"  "Occupation_Cleric."
## [7] "Occupation_Exec."   "Occupation_Prof."   "Occupation_Sales"
```

Coversion to factors:

```
data_df[columns_to_convert] <- lapply(data_df[columns_to_convert], as.factor)
```

```
str(data_df)
```

```
## spc_tbl_ [151,133 x 19] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ marital_status    : Factor w/ 2 levels "1","2": 1 2 2 1 2 2 1 2 2 2 ...
##  $ household_size    : num [1:151133] 2 2 2 2 2 2 2 2 2 2 ...
##  $ yrs_of_residence  : num [1:151133] 4 4 4 4 4 4 4 4 4 4 ...
##  $ Annual_Salary     : num [1:151133] 620 250 394 735 386 ...
##  $ Months_Annual     : num [1:151133] 2 3 4 1 6 3 18 2 4 8 ...
##  $ FRS.Contribution  : num [1:151133] 617 223 501 433 949 ...
##  $ Net_Salary        : num [1:151133] 502 468 514 562 666 ...
##  $ Net_months        : num [1:151133] 48 57 49 35 132 97 179 45 67 63 ...
##  $ Gross_Salary      : num [1:151133] 25.5 932.1 968.4 469.6 850.8 ...
##  $ Gross_Months      : num [1:151133] 46 56 48 34 128 95 173 44 65 62 ...
##  $ Qualify           : Factor w/ 2 levels "0","1": 1 1 1 1 2 2 2 1 1 1 ...
##  $ Education_Bach.   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Education_HS_grad : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Education_Masters : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Occupation_Cleric.: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Occupation_Exec.  : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Occupation_Prof.  : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
##  $ Occupation_Sales  : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ age               : num [1:151133] 48 60 82 47 75 74 78 46 75 76 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   marital_status = col_double(),
##   ..   household_size = col_double(),
##   ..   yrs_of_residence = col_double(),
##   ..   Annual_Salary = col_double(),
##   ..   Months_Annual = col_double(),
##   ..   FRS.Contribution = col_double(),
##   ..   Net_Salary = col_double(),
##   ..   Net_months = col_double(),
##   ..   Gross_Salary = col_double(),
##   ..   Gross_Months = col_double(),
##   ..   Qualify = col_double(),
##   ..   Education_Bach. = col_double(),
##   ..   `Education_HS-grad` = col_double(),
##   ..   Education_Masters = col_double(),
##   ..   Occupation_Cleric. = col_double(),
##   ..   Occupation_Exec. = col_double(),
##   ..   Occupation_Prof. = col_double(),
##   ..   Occupation_Sales = col_double(),
```

```
##   ..   age = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

check list

Before procceeding with the ml algorithm a check list has been created to ensure that the data is ready for the model:

1. check for missing values:

```
sum(is.na(data_df))
```

```
## [1] 0
```

there are no missing values.

2. Scaling Columns:

View the min and max values for each column:

```
min_max_df <- data.frame(
  Min = c(
    min(data_df$yrs_of_residence, na.rm = TRUE),
    min(data_df$Annual_Salary, na.rm = TRUE),
    min(as.numeric(as.character(data_df$Months_Annual)), na.rm = TRUE),
    min(data_df$FRS.Contribution, na.rm = TRUE),
    min(data_df$Net_Salary, na.rm = TRUE),
    min(data_df$Net_months, na.rm = TRUE),
    min(data_df$Gross_Salary, na.rm = TRUE),
    min(data_df$Gross_Months, na.rm = TRUE),
    min(as.numeric(as.character(data_df$household_size)), na.rm = TRUE),
    min(data_df$age, na.rm = TRUE)
  ),
  Max = c(
    max(data_df$yrs_of_residence, na.rm = TRUE),
    max(data_df$Annual_Salary, na.rm = TRUE),
    max(as.numeric(as.character(data_df$Months_Annual)), na.rm = TRUE),
    max(data_df$FRS.Contribution, na.rm = TRUE),
    max(data_df$Net_Salary, na.rm = TRUE),
    max(data_df$Net_months, na.rm = TRUE),
    max(data_df$Gross_Salary, na.rm = TRUE),
    max(data_df$Gross_Months, na.rm = TRUE),
    max(as.numeric(as.character(data_df$household_size)), na.rm = TRUE),
    max(data_df$age, na.rm = TRUE)
  ),
  row.names = c(
    "yrs_of_residence",
    "Annual_Salary",
    "Months_Annual",
    "FRS.Contribution",
    "Net_Salary",
    "Net_months",
    "Gross_Salary",
    "Gross_Months",
```

```
    "household_size",
    "age"
  )
)

# View the table
print(min_max_df)
```

```
##                    Min    Max
## yrs_of_residence  2.00   5.00
## Annual_Salary     0.00 999.98
## Months_Annual     1.00  48.00
## FRS.Contribution  0.08 999.98
## Net_Salary        0.01 999.97
## Net_months        1.00 322.00
## Gross_Salary      0.00 999.99
## Gross_Months      1.00 322.00
## household_size    2.00   3.00
## age              34.00 107.00
```

Columns to scale The following columns will be scaled due to their continous nature and large range between values: - Annual_Salary - FRS.Contribution - Net_Salary - Net_months - Gross_Salary - Gross_Months

Distribution The distribution of these columns will be viewed in-order to determine the best normalization function:

```
columns_to_distr <- c("Annual_Salary", "FRS.Contribution", "Net_Salary",
                      "Net_months", "Gross_Salary", "Gross_Months")
```

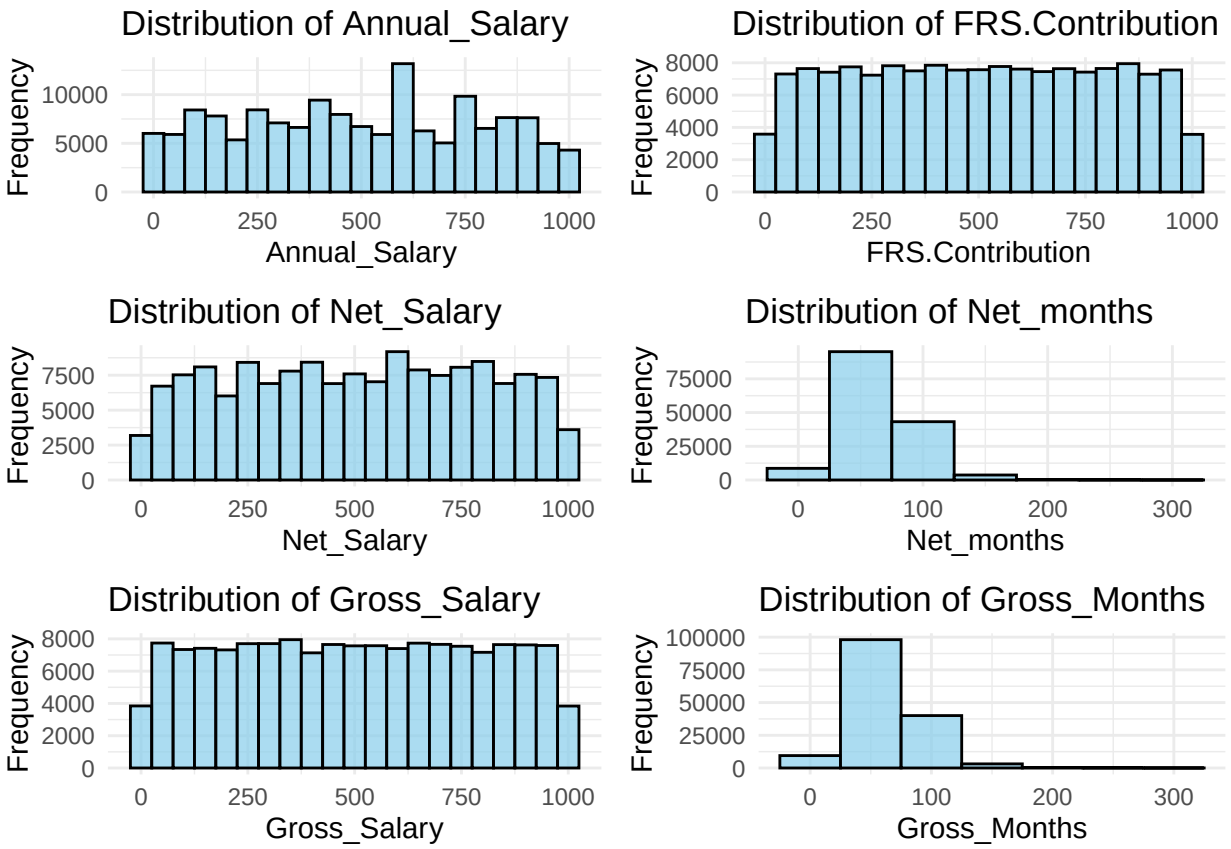
```
# folr loop to loop through the above columns:
plot_list <- list()

for (column in columns_to_distr) {
  p <- ggplot(data_df, aes_string(x = column)) +
    geom_histogram(binwidth = 50, fill = "skyblue", color = "black", alpha = 0.7) +
    labs(title = paste("Distribution of", column), x = column, y = "Frequency") +
    theme_minimal()

  plot_list[[column]] <- p
}
```

```
## Warning: `aes_string()` was deprecated in ggplot2 3.0.0.
## i Please use tidy evaluation idioms with `aes()`.
## i See also `vignette("ggplot2-in-packages")` for more information.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```r
grid.arrange(grobs = plot_list, ncol = 2)
```



Types of distributions in the dataset:

- Uniform Distribution: The following columns approximate a uniform distribution:
  - FRS.Contribution
  - Gross_Salary
- Skewed Distribution (Right skewed)
  - Gross_Months
  - Net_months
- Unidentified distribution (min-max scaling)
  - Net_Salary
  - Annual_Salary

```r
# Unidentified distribution
min_max_scaling <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}

# Z-Score Standardization Function (uniform distribution)
z_score_standardization <- function(x) {
  return((x - mean(x)) / sd(x))
}

# Log Function (skewed data)
log_transformation <- function(x) {
  return(log(x + 1))  # Adding 1 to avoid log(0)
```

```
}
```

Scale the following functions:

```
data_df <- data_df %>%
  mutate(
    FRS.Contribution = min_max_scaling(FRS.Contribution),
    Gross_Salary = min_max_scaling(Gross_Salary),
    Gross_Months = z_score_standardization(Gross_Months),
    Net_months = z_score_standardization(Net_months),
    Net_Salary = min_max_scaling(Net_Salary),
    Annual_Salary = min_max_scaling(Annual_Salary)
  )
```

View updated scaled values:

```
min_max_df <- data.frame(
  Min = c(
    min(data_df$yrs_of_residence, na.rm = TRUE),
    min(data_df$Annual_Salary, na.rm = TRUE),
    min(as.numeric(as.character(data_df$Months_Annual)), na.rm = TRUE),
    min(data_df$FRS.Contribution, na.rm = TRUE),
    min(data_df$Net_Salary, na.rm = TRUE),
    min(data_df$Net_months, na.rm = TRUE),
    min(data_df$Gross_Salary, na.rm = TRUE),
    min(data_df$Gross_Months, na.rm = TRUE),
    min(as.numeric(as.character(data_df$household_size)), na.rm = TRUE),
    min(data_df$age, na.rm = TRUE)
  ),
  Max = c(
    max(data_df$yrs_of_residence, na.rm = TRUE),
    max(data_df$Annual_Salary, na.rm = TRUE),
    max(as.numeric(as.character(data_df$Months_Annual)), na.rm = TRUE),
    max(data_df$FRS.Contribution, na.rm = TRUE),
    max(data_df$Net_Salary, na.rm = TRUE),
    max(data_df$Net_months, na.rm = TRUE),
    max(data_df$Gross_Salary, na.rm = TRUE),
    max(data_df$Gross_Months, na.rm = TRUE),
    max(as.numeric(as.character(data_df$household_size)), na.rm = TRUE),
    max(data_df$age, na.rm = TRUE)
  ),
  row.names = c(
    "yrs_of_residence",
    "Annual_Salary",
    "Months_Annual",
    "FRS.Contribution",
    "Net_Salary",
    "Net_months",
    "Gross_Salary",
    "Gross_Months",
    "household_size",
    "age"
  )
)
```

```r
# View the table
print(min_max_df)
```

```
##                         Min        Max
## yrs_of_residence   2.000000   5.000000
## Annual_Salary      0.000000   1.000000
## Months_Annual      1.000000  48.000000
## FRS.Contribution   0.000000   1.000000
## Net_Salary         0.000000   1.000000
## Net_months        -2.111564   8.783532
## Gross_Salary       0.000000   1.000000
## Gross_Months      -2.102705   9.051311
## household_size     2.000000   3.000000
## age               34.000000 107.000000
```

Split the data set for the model

- Training
- Testing
- Validation

```r
set.seed(123)
total_rows <- nrow(data_df)

#split data 70-30
train_indices <- sample(1:total_rows, 0.7 * total_rows)
train_data <- data_df[train_indices, ]

remaining_indices <- setdiff(1:total_rows, train_indices)

#testing and validation will each make up 15%
validation_indices <- sample(remaining_indices, 0.5 * length(remaining_indices))

test_indices <- setdiff(remaining_indices, validation_indices)

validation_data <- data_df[validation_indices, ]
test_data <- data_df[test_indices, ]
```

```r
cat("Training data size:", nrow(train_data), "\n")
```

```
## Training data size: 105793
```

```r
cat("Validation data size:", nrow(validation_data), "\n")
```

```
## Validation data size: 22670
```

```r
cat("Testing data size:", nrow(test_data), "\n")
```

```
## Testing data size: 22670
```

Dataset successfully split!

Initialise the rf model

```r
library(randomForest)
```

```
## randomForest 4.7-1.2
```

```
## 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:gridExtra':
##
##     combine
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(caret)
```

```
## Loading required package: lattice
```

Define a formula for the rf classifier:

```
formula <- Qualify ~ marital_status + household_size + yrs_of_residence +
  Annual_Salary + Months_Annual + FRS.Contribution +
  Net_Salary + Net_months + Gross_Salary + Gross_Months +
  Education_Bach. + Education_HS_grad + Education_Masters +
  Occupation_Cleric. + Occupation_Exec. + Occupation_Prof. +
  Occupation_Sales + age
```

Train the rf-model:

```
model <- randomForest(formula, data = train_data)
```

Test model performance using the Test set

Make predictions to evaluate the performance of the model.

```
predictions <- predict(model, newdata = test_data)
```

Create a Confusion Matrix

The confusion matrix here is utilized as a performance measurement tool. It will be used as one of the tools that determine how well the classification model performs.

```
# Create confusion matrix
confusion_matrix <- confusionMatrix(predictions, test_data$Qualify)
print(confusion_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 13529     0
##          1     0  9141
##
##                Accuracy : 1
##                  95% CI : (0.9998, 1)
##     No Information Rate : 0.5968
```

```
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0000
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##              Prevalence : 0.5968
##          Detection Rate : 0.5968
##    Detection Prevalence : 0.5968
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 0
##
```

Tabulate some of the output from the confusion matrix:

```r
# Create confusion matrix
confusion_matrix <- table(
  Actual = test_data$Qualify,
  Predicted = predictions
)

formatted_confusion_matrix <- matrix(0, nrow = 2, ncol = 2)
rownames(formatted_confusion_matrix) <- c("Actual Positive (Yes)", "Actual Negative (No)")
colnames(formatted_confusion_matrix) <- c("Predicted Positive (Yes)", "Predicted Negative (No)")

# Fill
formatted_confusion_matrix[1, 1] <- confusion_matrix["1", "1"]
formatted_confusion_matrix[1, 2] <- confusion_matrix["1", "0"]
formatted_confusion_matrix[2, 1] <- confusion_matrix["0", "1"]
formatted_confusion_matrix[2, 2] <- confusion_matrix["0", "0"]

# Convert row and column names to "Yes" and "No"
rownames(formatted_confusion_matrix) <- c("Actual Positive (Yes)", "Actual Negative (No)")
colnames(formatted_confusion_matrix) <- c("Predicted Positive (Yes)", "Predicted Negative (No)")

# View the formatted confusion matrix
print(formatted_confusion_matrix)
```

```
##                       Predicted Positive (Yes) Predicted Negative (No)
## Actual Positive (Yes)                     9141                       0
## Actual Negative (No)                         0                   13529
```

Interpretation of results

- True positives: These are areas where the model correctly predicted yes (9141).
- Under predicted negatives (True negatives), we see that the model did not predict negative values as positive. Just from these two readings we see that the model did not miss classify.It has a good recall and precision.

Interpreting the metrices from the confusion report

- Accuracy (100%): The model correctly identifies all records in the data set.
- Confidence Interval (95%): With 95% confidence, it can be stated that the true accuracy of the model falls between 99.98% and 100%.
- P-value (<2.2eˆ-16):
  - This low p-values suggests that the accuracy of the model is better than what we would get from random chance.

The models metrices so far are excellent which is rather suspicious. Hence, before proceeding to the testing; the balanace of the classes will be checked with the aim of identifying if the classes are balanced.

Checking for class imbalance

Class imbalance can lead to high accuracy scores of one class dominates another.If there is a dominating class sampling techniques will be required (under-sampling or over-sampling):

```
class_counts <- table(train_data$Qualify)
```

```
ggplot(data = as.data.frame(class_counts), aes(x = Var1, y = Freq)) +
  geom_bar(stat = "identity", fill = "yellow") +
  labs(title = "Class Distribution in Training Data",
       x = "Class",
       y = "Count") +
  theme_minimal()
```


```
class_percentages <- prop.table(class_counts) * 100
class_percentages
```

```
##
##      0      1
## 59.969 40.031
```

The difference in the classes is not significantly large as to require balancing techniques.

Validate the models performace using the Validation set

```
validation_predictions <- predict(model, newdata = validation_data)

validation_confusion_matrix <- table(Actual = validation_data$Qualify, Predicted = validation_predictio
```

Performance Metrics:

```
accuracy_val <- sum(diag(validation_confusion_matrix)) / sum(validation_confusion_matrix)
sensitivity_val <- validation_confusion_matrix[2, 2] / sum(validation_confusion_matrix[2, ])
specificity_val <- validation_confusion_matrix[1, 1] / sum(validation_confusion_matrix[1, ])
precision_val <- validation_confusion_matrix[2, 2] / sum(validation_confusion_matrix[, 2])
recall_val <- sensitivity_val
f1_score_val <- 2 * (precision_val * recall_val) / (precision_val + recall_val)
```

```
cat("Validation Performance Metrics:\n")
```

```
## Validation Performance Metrics:
```

```
cat("Accuracy:", round(accuracy_val, 4), "\n")
```

```
## Accuracy: 1
```

```
cat("Sensitivity (Recall):", round(sensitivity_val, 4), "\n")
```

```
## Sensitivity (Recall): 1
```

```
cat("Specificity:", round(specificity_val, 4), "\n")
```

```
## Specificity: 1
```

```
cat("Precision:", round(precision_val, 4), "\n")
```

```
## Precision: 1
```

```
cat("F1 Score:", round(f1_score_val, 4), "\n")
```

```
## F1 Score: 1
```

They are all 100%, this could suggest over-fitting. hence the next stage is to drop the column used to generate the target column.

drop the Net_Salary column

The net salary column was used to identify qualifying customers; hence dropping it could fix the over-fitting problem. as the suspicion is that over-fitting is due to this feature causing leakage.

```
train_data <- train_data[, !names(train_data) %in% "Net_Salary"]
validation_data <- validation_data[, !names(validation_data) %in% "Net_Salary"]
test_data <- test_data[, !names(test_data) %in% "Net_Salary"]
```

Retrain the model and predict:

```
model_retrained <- randomForest(Qualify ~ ., data = train_data)
```

```r
# Validate the model again using the validation set
validation_predictions_retrained <- predict(model_retrained, newdata = validation_data)
```

```r
validation_confusion_matrix_retrained <- table(Actual = validation_data$Qualify, Predicted = validation_
```

performance metrics:

```r
accuracy_retrained <- sum(diag(validation_confusion_matrix_retrained)) / sum(validation_confusion_matri
sensitivity_retrained <- validation_confusion_matrix_retrained[2, 2] / sum(validation_confusion_matrix_
specificity_retrained <- validation_confusion_matrix_retrained[1, 1] / sum(validation_confusion_matrix_
precision_retrained <- validation_confusion_matrix_retrained[2, 2] / sum(validation_confusion_matrix_re
recall_retrained <- sensitivity_retrained
f1_score_retrained <- 2 * (precision_retrained * recall_retrained) / (precision_retrained + recall_retra
```

```r
cat("Accuracy:", round(accuracy_retrained * 100, 2), "%\n")
```

```
## Accuracy: 87.87 %
```

```r
cat("Sensitivity (Recall):", round(sensitivity_retrained * 100, 2), "%\n")
```

```
## Sensitivity (Recall): 81.3 %
```

```r
cat("Specificity:", round(specificity_retrained * 100, 2), "%\n")
```

```
## Specificity: 92.25 %
```

```r
cat("Precision:", round(precision_retrained * 100, 2), "%\n")
```

```
## Precision: 87.51 %
```

```r
cat("F1 Score:", round(f1_score_retrained * 100, 2), "%\n")
```

```
## F1 Score: 84.29 %
```

Metrics Interpretation

Accuracy (88.04%): - The model correctly predicted 87% of the total records in the validation set. The classes are slightly imbalanced but not significantly as to highly affect the accuracy score.

- Recall (Sensitivity) (81.5%):
  - The model's ability to correctly classify positive (1) records. The model correctly identified 81% of True positives.
  - Which means 18.5% of true positives we predicted to be negative.
  - The model correctly identified 81.5% of customers who qualify for the service.
- Specificity (92.25%):
  - The models ability to classify true negatives.
  - the model successfully identified 95% of applicants who do not qualify for the service.
- Precision (87.76%):
  - This is the measure of the accuracy of positive predictions.
- F1 Score (84.51%): This is the true measure of the performance of the model. it is the harmonic mean of precision and sensitivity. An 84% F1-Score indicates a solid model performance in predicting customer who qualify for the service and those who do not.

The RF model was successfully implemented.

##ft importance Plot the feature importance to close off this section:

```r
# Extract feature importance
importance_values <- importance(model_retrained)
```

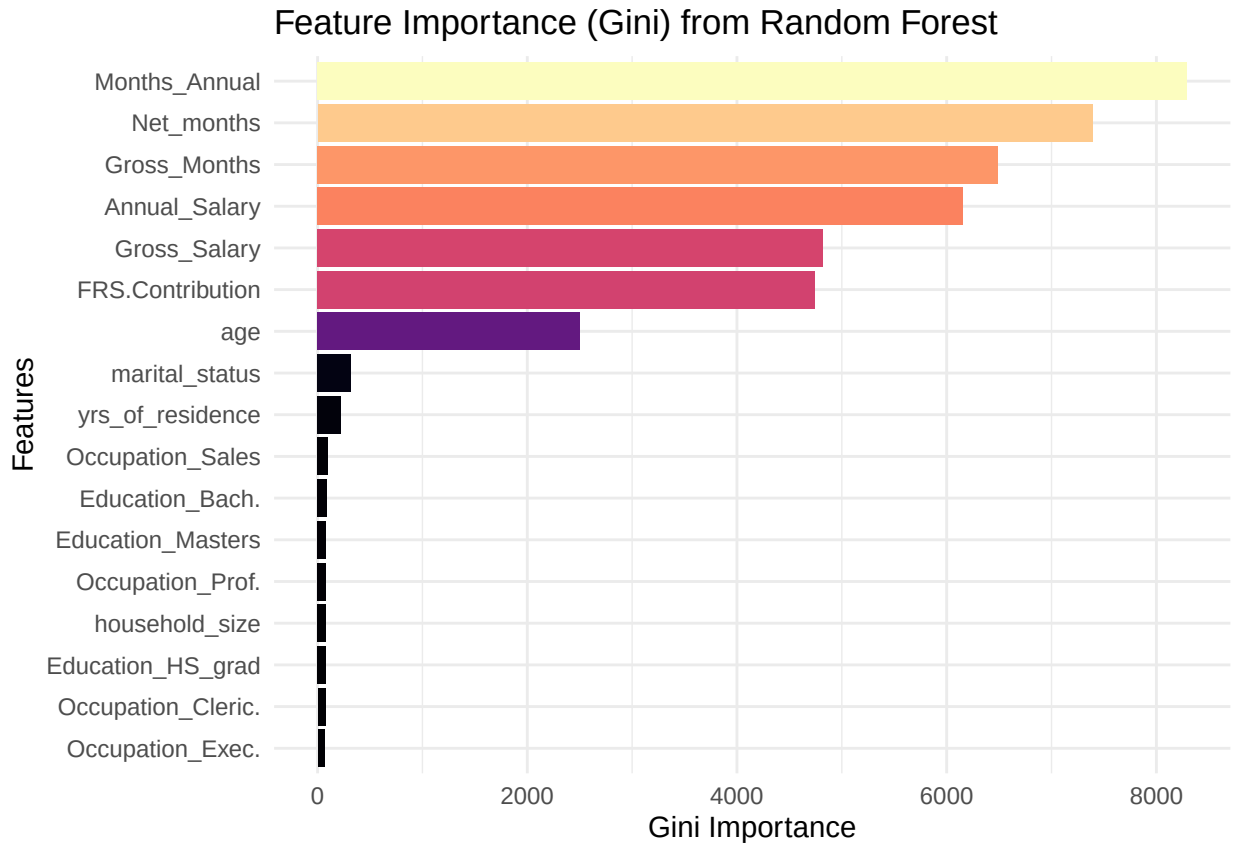
Gini Importance:

```
importance_df <- data.frame(
  Feature = rownames(importance_values),
  GiniImportance = importance_values[, "MeanDecreaseGini"]
)
```

```
#colour map for plot
library(viridis)
```

```
## Loading required package: viridisLite
```

```
ggplot(importance_df, aes(x = reorder(Feature, GiniImportance), y = GiniImportance, fill = GiniImportan
  geom_bar(stat = "identity") +
  coord_flip() +  # Flip for better readability
  labs(
    title = "Feature Importance (Gini) from Random Forest",
    x = "Features",
    y = "Gini Importance"
  ) +
  scale_fill_viridis(
    option = "magma",  # Use the magma colormap
    name = "Importance"
  ) +
  theme_minimal() +
  theme(legend.position = "none")
```



```r
saveRDS(model_retrained, "retrained_rf_model.rds")
```

The Random forest model has been successfully saved.

save final form of data into a csv

```r
print(names(data_df))
```

```
##  [1] "marital_status"     "household_size"     "yrs_of_residence"
##  [4] "Annual_Salary"      "Months_Annual"      "FRS.Contribution"
##  [7] "Net_Salary"         "Net_months"         "Gross_Salary"
## [10] "Gross_Months"       "Qualify"            "Education_Bach."
## [13] "Education_HS_grad"  "Education_Masters"  "Occupation_Cleric."
## [16] "Occupation_Exec."   "Occupation_Prof."   "Occupation_Sales"
## [19] "age"
```