The University of Texas at Dallas

Naveen Jindal School of Management

Fall 2022

BA WITH R – BUAN 6356.006

Under the Guidance of: **Prof. Zhe Zhang**

CAR INSURANCE CLAIM CLASSIFIER

GROUP - 12

MEMBERS:

Manusha Medarametla

Jahnani Nagarajan Sivakumar

Kirthika Kulandaivel Senthilkumar

Krishna Apurva

INDEX

CONTENT | **PAGE NO.**

INTRODUCTION

The insurance industry is transforming data analytics and predictive modeling, especially car insurance. The "Car Insurance Claim Prediction" aims to predict if policyholders will file a claim in the next six months by analyzing a comprehensive dataset. This insight will help insurance companies refine their risk assessment and pricing strategies. Advanced analytics and machine learning can revolutionize managing risk in the car insurance industry. The project aims to develop an accurate predictive model using policyholder attributes for data-informed decision-making.

PROBLEM STATEMENT

The "**Car Insurance Claim Prediction**" aims to predict if policyholders will **file an insurance claim in the next six months** by analyzing a comprehensive dataset. This insight will help insurance companies refine their **risk assessment and pricing strategies.**

PROJECT OBJECTIVES:

- To collect a comprehensive dataset of policy holder's personal and vehicle insurance details
- To explore the dataset and understand the predictor & outcome variables and observation records within the dataset.
- To gain a solid grasp of supervised learning data analytics methodologies based on the target outcome variable.
- To perform data pre-processing techniques to replace null values with data imputation, standardize the numerical values, and encode any categorical values.
- To build a predictive model to train the dataset using selective predictor variables to forecast whether policyholders will file a car insurance claim within the next six months.
- To validate the prediction model accuracy using the evaluation/validation dataset and techniques.
- To analyze and gather data-driven information from the model and optimize for accuracy.

PROJECT MOTIVATION

The "Car Insurance Claim Prediction" project is driven by the evolving landscape of data analytics and predictive modeling in the insurance sector, particularly car insurance. Analyzing a comprehensive dataset, the project aims to predict policyholders' likelihood of filing a claim within six months. This predictive insight will revolutionize risk assessment and pricing strategies, empowering insurance companies to manage risk in the industry proactively.
The primary objective is to craft a highly accurate predictive model using policyholder attributes. This model will facilitate data-informed decision-making, allowing insurance firms to pre-emptively address risks and optimize operational efficiency.

DATA DESCRIPTION

Data mining techniques have a wide range of applications across various domains. For our project, we have chosen to work on the problem of predicting car insurance claims. We have collected 43 input attributes such as policy tenure, age of the car, model, segment, fuel type, etc. The target variable is whether a claim is made, represented by the binary variable **'is_claim.'** This is a classification problem, and we have used four different algorithms - Logistic Regression, Decision Trees, Random Forests, and Neural Networks - to predict the loan status.

The dataset we have taken is from Kaggle, and it has over 43 input attributes that help us understand the Claim prediction. We have plotted several graphs as part of our exploratory data analysis to understand our data better.

Variable	Description
policy_id	Unique identifier of the policyholder
policy_tenure	Time period of the policy
age_of_car	Normalized age of the car in years
age_of_policyholder	Normalized age of policyholder in years
area_cluster	Area cluster of the policyholder
population density	Population density of the city (Policyholder City)
make	Encoded Manufacturer/company of the car
segment	Segment of the car (A/ B1/ B2/ C1/ C2)
model	Encoded name of the car
fuel_type	Type of fuel used by the car
max_torque	Maximum Torque generated by the car (Nm@rpm)
max_power	Maximum Power generated by the car (bhp@rpm)
engine_type	Type of engine used in the car
airbags	Number of airbags installed in the car
is_esc	Boolean flag indicating whether Electronic Stability Control (ESC) is present in the car or not.
is_adjustable_steering	Boolean flag indicating whether the steering wheel of the car is adjustable or not.
is_tpms	Boolean flag indicating whether Tyre Pressure Monitoring System (TPMS) is present in the car or not.
is_parking_sensors	Boolean flag indicating whether parking sensors are present in the car or not.
is_parking_camera	Boolean flag indicating whether the parking camera is present in the car or not.
rear_brakes_type	Type of brakes used in the rear of the car
displacement	Engine displacement of the car (cc)
cylinder	Number of cylinders present in the engine of the car
transmission_type	Transmission type of the car
gear_box	Number of gears in the car
steering_type	Type of the power steering present in the car
turning_radius	The space a vehicle needs to make a certain turn (Meters)
length	Length of the car (Millimetre)
width	Width of the car (Millimetre)
height	Height of the car (Millimetre)
gross_weight	The maximum allowable weight of the fully-loaded car, including passengers, cargo and equipment (Kg)
is_front_fog_lights	Boolean flag indicating whether front fog lights are available in the car or not.
is_rear_window_wiper	Boolean flag indicating whether the rear window wiper is available in the car or not.
is_rear_window_washer	Boolean flag indicating whether the rear window washer is available in the car or not.
is_rear_window_defogger	Boolean flag indicating whether rear window defogger is available in the car or not.
is_brake_assist	Boolean flag indicating whether the brake assistance feature is available in the car or not.
is_power_door_lock	Boolean flag indicating whether a power door lock is available in the car or not.
is_central_locking	Boolean flag indicating whether the central locking feature is available in the car or not.
is_power_steering	Boolean flag indicating whether power steering is available in the car or not.
is_driver_seat_height_adjustable	Boolean flag indicating whether the height of the driver seat is adjustable or not.
is_day_night_rear_view_mirror	Boolean flag indicating whether day & night rearview mirror is present in the car or not.
is_ecw	Boolean flag indicating whether Engine Check Warning (ECW) is available in the car or not.
is_speed_alert	Boolean flag indicating whether the speed alert system is available in the car or not.
ncap_rating	Safety rating given by NCAP (out of 5)
is_claim	Outcome: Boolean flag indicating whether the policyholder file a claim in the next 6 months or not.

EXPLORATORY DATA ANALYSIS

During the exploratory data analysis (EDA) phase, we thoroughly examined the dataset, paying close attention to different aspects. Our goal was to extract valuable insights from the data and ensure it was well-prepared for modeling purposes.

[1] Verifying the summary of the structure of the dataset to understand the data values

```
> # Summary of data types
> str(df.train)
'data.frame':	58592 obs. of  44 variables:
 $ policy_id                       : chr  "ID00001" "ID00002" "ID00003" "ID00004" ...
 $ policy_tenure                   : num  0.516 0.673 0.841 0.9 0.596 ...
 $ age_of_car                      : num  0.05 0.02 0.02 0.11 0.11 0.07 0.16 0.14 0.07 0.04 ...
 $ age_of_policyholder             : num  0.644 0.375 0.385 0.433 0.635 ...
 $ area_cluster                    : chr  "C1" "C2" "C3" "C4" ...
 $ population_density              : int  4990 27003 4076 21622 34738 13051 6112 8794 6112 17804 ...
 $ make                            : int  1 1 1 1 2 3 4 1 3 1 ...
 $ segment                         : chr  "A" "A" "A" "C1" ...
 $ model                           : chr  "M1" "M1" "M1" "M2" ...
 $ fuel_type                       : chr  "CNG" "CNG" "CNG" "Petrol" ...
 $ max_torque                      : chr  "60Nm@3500rpm" "60Nm@3500rpm" "60Nm@3500rpm" "113Nm@4400rpm" ...
 $ max_power                       : chr  "40.36bhp@6000rpm" "40.36bhp@6000rpm" "40.36bhp@6000rpm" "88.50bhp@6000rpm" ...
 $ engine_type                     : chr  "F8D Petrol Engine" "F8D Petrol Engine" "F8D Petrol Engine" "1.2 L K12N Dualjet" ...
 $ airbags                         : int  2 2 2 2 2 6 2 2 6 6 ...
 $ is_esc                          : chr  "No" "No" "No" "Yes" ...
 $ is_adjustable_steering          : chr  "No" "No" "No" "Yes" ...
 $ is_tpms                         : chr  "No" "No" "No" "No" ...
 $ is_parking_sensors              : chr  "Yes" "Yes" "Yes" "Yes" ...
 $ is_parking_camera               : chr  "No" "No" "No" "Yes" ...
 $ rear_brakes_type                : chr  "Drum" "Drum" "Drum" "Drum" ...
 $ displacement                    : int  796 796 796 1197 999 1493 1497 1197 1493 1197 ...
 $ cylinder                        : int  3 3 3 4 3 4 4 4 4 4 ...
 $ transmission_type               : chr  "Manual" "Manual" "Manual" "Automatic" ...
 $ gear_box                        : int  5 5 5 5 5 6 5 5 6 5 ...
 $ steering_type                   : chr  "Power" "Power" "Power" "Electric" ...
 $ turning_radius                  : num  4.6 4.6 4.6 4.8 5 5.2 5 4.8 5.2 4.85 ...
 $ length                          : int  3445 3445 3445 3995 3731 4300 3990 3845 4300 3990 ...
 $ width                           : int  1515 1515 1515 1735 1579 1790 1755 1735 1790 1745 ...
 $ height                          : int  1475 1475 1475 1515 1490 1635 1523 1530 1635 1500 ...
 $ gross_weight                    : int  1185 1185 1185 1335 1155 1720 1490 1335 1720 1410 ...
 $ is_front_fog_lights             : chr  "No" "No" "No" "Yes" ...
 $ is_rear_window_wiper            : chr  "No" "No" "No" "No" ...
 $ is_rear_window_washer           : chr  "No" "No" "No" "No" ...
 $ is_rear_window_defogger         : chr  "No" "No" "No" "Yes" ...
 $ is_brake_assist                 : chr  "No" "No" "No" "Yes" ...
 $ is_power_door_locks             : chr  "No" "No" "No" "Yes" ...
 $ is_central_locking              : chr  "No" "No" "No" "Yes" ...
 $ is_power_steering               : chr  "Yes" "Yes" "Yes" "Yes" ...
 $ is_driver_seat_height_adjustable: chr  "No" "No" "No" "Yes" ...
 $ is_day_night_rear_view_mirror   : chr  "No" "No" "No" "Yes" ...
 $ is_ecw                          : chr  "No" "No" "No" "Yes" ...
 $ is_speed_alert                  : chr  "Yes" "Yes" "Yes" "Yes" ...
 $ ncap_rating                     : int  0 0 0 2 2 3 5 2 3 0 ...
 $ is_claim                        : int  0 0 0 0 0 0 0 0 0 0 ...
```

R CODE –

```
# Summary of data types
      str(df.train)
```

[2] Next, we verify that we have any NA value or NULL value in the dataset.

```
> na_count <- colSums(is.na(df.train))
> na_count
                     policy_id                 policy_tenure                    age_of_car
                             0                             0                             0
            age_of_policyholder                  area_cluster            population_density
                             0                             0                             0
                           make                       segment                         model
                             0                             0                             0
                      fuel_type                    max_torque                     max_power
                             0                             0                             0
                    engine_type                       airbags                        is_esc
                             0                             0                             0
         is_adjustable_steering                       is_tpms            is_parking_sensors
                             0                             0                             0
               is_parking_camera              rear_brakes_type                  displacement
                             0                             0                             0
                       cylinder             transmission_type                      gear_box
                             0                             0                             0
                  steering_type                turning_radius                        length
                             0                             0                             0
                          width                        height                  gross_weight
                             0                             0                             0
            is_front_fog_lights            is_rear_window_wiper          is_rear_window_washer
                             0                             0                             0
         is_rear_window_defogger               is_brake_assist           is_power_door_locks
                             0                             0                             0
              is_central_locking             is_power_steering is_driver_seat_height_adjustable
                             0                             0                             0
 is_day_night_rear_view_mirror                        is_ecw                is_speed_alert
                             0                             0                             0
                    ncap_rating                      is_claim
                             0                             0
```

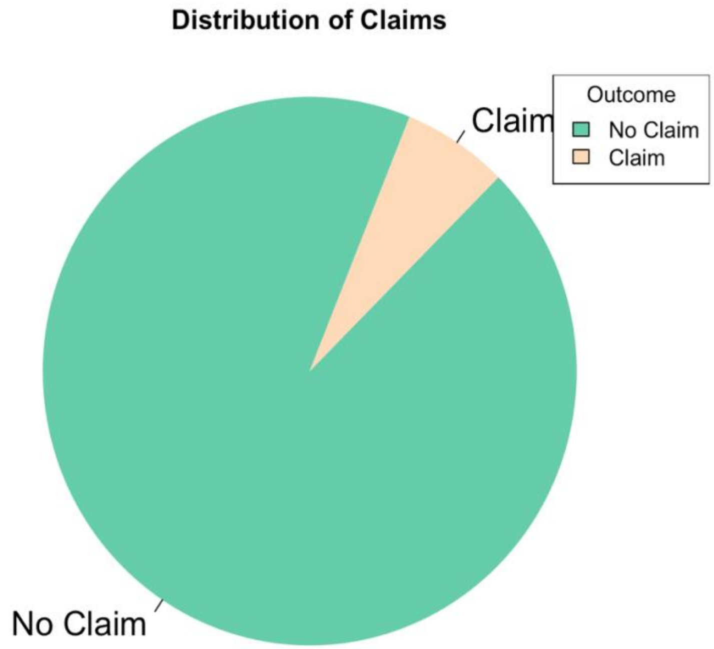
<u>R CODE –</u>

```
# Count the number of missing values (NA) in each column of the dataframe 'df.train'
    na_count <- colSums(is.na(df.train))
    na_count
```

[3] A pie chart is generated by the code to display the distribution of instances belonging to the 'No Claim' and 'Claim' categories. This pie chart helps to reveal the class imbalance in the target variable 'is_claim.' The legend and colors used in the chart accurately depict the two categories, emphasizing the necessity of addressing this imbalance while constructing the model to achieve better predictive accuracy.

The chart shows the percentage of No claims for car insurance and claims for car insurance.

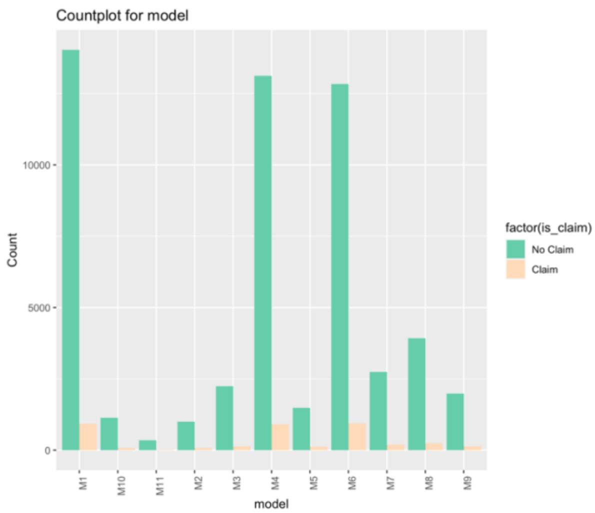
- The Percentage of No claims for car insurance is **93.4%**
- The percentage of claims for car insurance is **6.4%**.

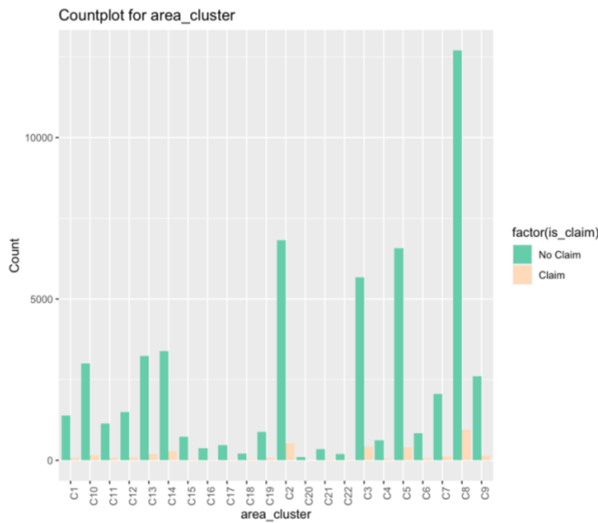


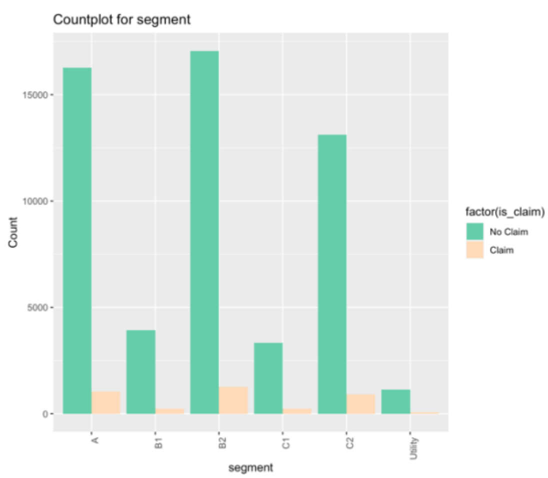
R CODE –

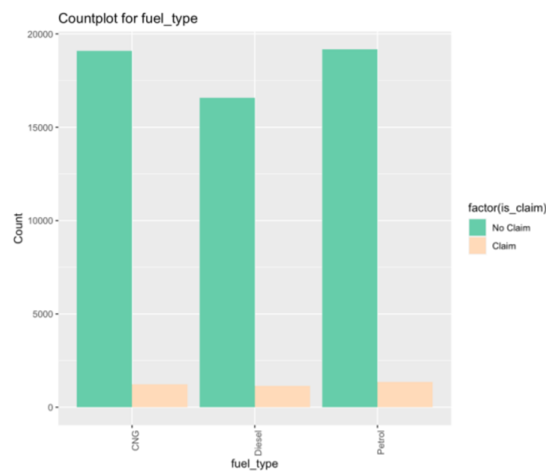
```
# Calculate counts of 'No Claim' and 'Claim'
    claim_counts <- table(df.train_filter$is_claim)

# Create a pie chart
    pie(claim_counts, labels = c('No Claim', 'Claim'), radius =
    1, col = c('mediumaquamarine', 'peachpuff'),
        init.angle = 45, clockwise = TRUE, border = NA, cex =
    1.5,
        main = 'Distribution of Claims')

# The target variable has imbalance data and we will address
this while building the model
# Add legend
  legend("topright", c('No Claim', 'Claim'), fill =
  c('mediumaquamarine', 'peachpuff'), title = 'Outcome')
```

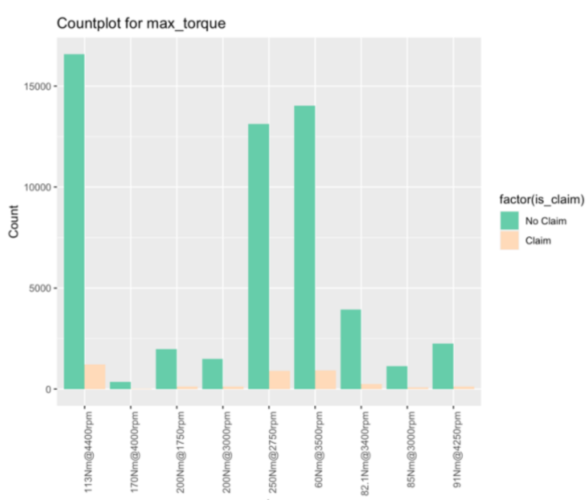
[4] The count plots showcasing the distribution of different categorical variables like 'area_cluster,' 'segment,' 'model,' 'fuel_type,' 'max_torque,' 'max_power,' and 'engine_type' concerning the 'No Claim' and 'Claim' classes. Each plot visualizes the count of occurrences for both classes, aiding in understanding the class distribution within these categorical features. These visualizations offer insights into potential correlations between categorical variables and the claim outcomes, providing a clear understanding of their impact on the target variable.

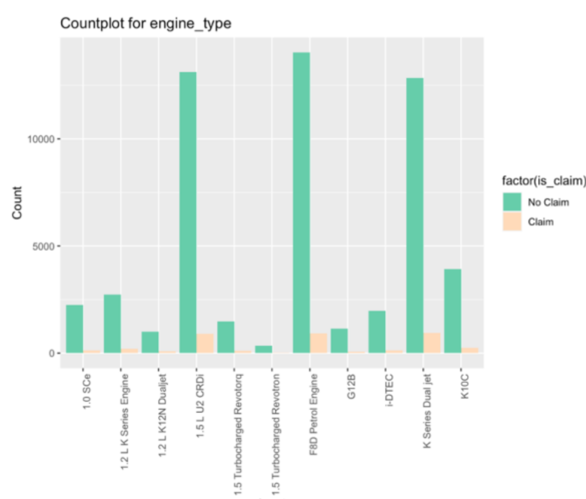


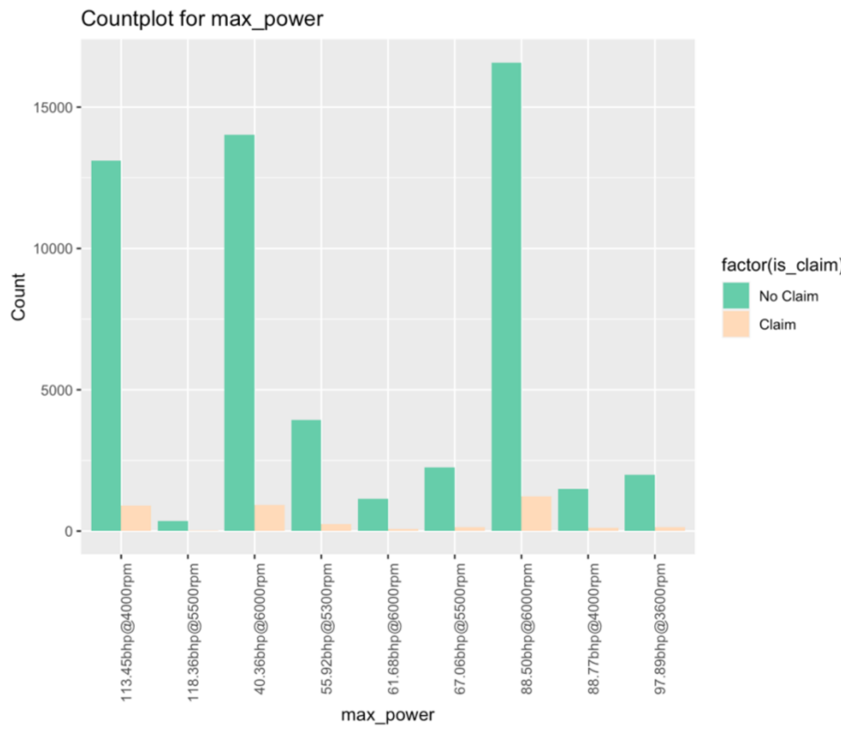












R CODE –

```
# Count plot for 'area_cluster'
    plot_area_cluster <- ggplot(df.train_filter, aes(x =
    area_cluster, fill = factor(is_claim))) +
      geom_bar(position = 'dodge') +
      labs(title = "Countplot for area_cluster", y = "Count") +
      scale_fill_manual(values = c("mediumaquamarine",
    "peachpuff"), labels = c("No Claim", "Claim")) +
      theme(axis.text.x = element_text(angle = 90, hjust = 1))

# Count plot for 'segment'
    plot_segment <- ggplot(df.train_filter, aes(x = segment,
    fill = factor(is_claim))) +
      geom_bar(position = 'dodge') +
      labs(title = "Countplot for segment", y = "Count") +
      scale_fill_manual(values = c("mediumaquamarine",
    "peachpuff"), labels = c("No Claim", "Claim")) +
      theme(axis.text.x = element_text(angle = 90, hjust = 1))

# Count plot for 'model'
    plot_model <- ggplot(df.train_filter, aes(x = model, fill =
    factor(is_claim))) +
```

```r
    geom_bar(position = 'dodge') +
    labs(title = "Countplot for model", y = "Count") +
    scale_fill_manual(values = c("mediumaquamarine",
  "peachpuff"), labels = c("No Claim", "Claim")) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))

# Count plot for 'fuel_type'
  plot_fuel_type <- ggplot(df.train_filter, aes(x =
  fuel_type, fill = factor(is_claim))) +
    geom_bar(position = 'dodge') +
    labs(title = "Countplot for fuel_type", y = "Count") +
    scale_fill_manual(values = c("mediumaquamarine",
  "peachpuff"), labels = c("No Claim", "Claim")) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))

# Count plot for 'max_torque'
  plot_max_torque <- ggplot(df.train_filter, aes(x =
  max_torque, fill = factor(is_claim))) +
    geom_bar(position = 'dodge') +
    labs(title = "Countplot for max_torque", y = "Count") +
    scale_fill_manual(values = c("mediumaquamarine",
  "peachpuff"), labels = c("No Claim", "Claim")) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))

# Count plot for 'max_power'
  plot_max_power <- ggplot(df.train_filter, aes(x =
  max_power, fill = factor(is_claim))) +
    geom_bar(position = 'dodge') +
    labs(title = "Countplot for max_power", y = "Count") +
    scale_fill_manual(values = c("mediumaquamarine",
  "peachpuff"), labels = c("No Claim", "Claim")) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))

# Count plot for 'engine_type'
  plot_engine_type <- ggplot(df.train_filter, aes(x =
  engine_type, fill = factor(is_claim))) +
    geom_bar(position = 'dodge') +
    labs(title = "Countplot for engine_type", y = "Count") +
    scale_fill_manual(values = c("mediumaquamarine",
  "peachpuff"), labels = c("No Claim", "Claim")) +
    theme(axis.text.x = element_text(angle = 90, hjust = 1))

# Count plot for 'is_speed_alert'
```

```r
plot_speed_alert <- ggplot(df.train_filter, aes(x =
is_speed_alert, fill = factor(is_claim))) +
  geom_bar(position = 'dodge') +
  labs(title = "Countplot for is_speed_alert", y = "Count")
+
  scale_fill_manual(values = c("mediumaquamarine",
"peachpuff"), labels = c("No Claim", "Claim")) +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))

# Display individual plots
plot_area_cluster
plot_segment
plot_model
plot_fuel_type
plot_max_torque
plot_max_power
plot_engine_type
plot_speed_alert
```

[5] As part of our data processing, we extract numerical values in Newton-meters (Nm) and revolutions per minute (rpm) from the 'max_torque' data. After that, we calculate the ratio of torque to rpm and add a new column named 'torque_to_rpm_ratio' to the dataset to consolidate this information. Similarly, we are doing for the 'max_power' data.

<u>R CODE –</u>

```r
##################### Convert Torque into Torque/rpm ###########

# Extract torque values from 'max_torque' column using regex
pattern
df.train_filter$torque <-
  str_extract(df.train_filter$max_torque,
  "\\d+\\.?\\d*(?=Nm)")
df.train_filter$rpm <- str_extract(df.train_filter$max_torque,
  "\\d+\\.?\\d*(?=rpm)")

# Convert 'torque' column from character to numeric
df.train_filter$torque <- as.numeric(df.train_filter$torque)
df.train_filter$rpm <- as.numeric(df.train_filter$rpm)

# Calculate torque to RPM ratio and add as a new column
df.train_filter$torque_to_rpm_ratio <- df.train_filter$torque /
  df.train_filter$rpm
```

```
#Remove max_torque
  cols_rmv <- c("max_torque","torque", "rpm")

  df.train_filter  <-  df.train_filter[,!(names(df.train_filter)
    %in% cols_rmv)]

#################### Convert Power into Power/rpm #############

# Extract torque values from 'max_power' column using regex pattern
  df.train_filter$power                                       <-
    str_extract(df.train_filter$max_power,
    "\\d+\\.?\\d*(?=bhp)")
  df.train_filter$rpm  <-  str_extract(df.train_filter$max_power,
    "\\d+\\.?\\d*(?=rpm)")

# Convert 'Power' column from character to numeric
  df.train_filter$power <- as.numeric(df.train_filter$power)
  df.train_filter$rpm <- as.numeric(df.train_filter$rpm)

# Calculate power to RPM ratio and add as a new column
  df.train_filter$power_to_rpm_ratio <- df.train_filter$power /
    df.train_filter$rpm

#Remove max_torque and max_power
  cols_rmv <- c("max_power", "power", "rpm")

  df.train_filter  <-  df.train_filter[,!(names(df.train_filter)
    %in% cols_rmv)]
```

[6] To handle binary categorical columns in the dataset 'df.train_filter', we first identified the columns that were prefixed with "is_" using a pattern-matching approach. After the identification process, we proceeded to convert the categorical values ("Yes" and "No") into numeric format by assigning 1 to "Yes" and 0 to "No" across all the identified 'is_' columns within the dataset.

R CODE –

```
##### Convert "is_" column's 'Yes' and 'No' into 1 and 0 #####

#  Get  column  names  that  contain  "is_"  in  the  dataframe
'df.train_filter'

  is_cols <- grep("^is_", names(df.train_filter), value = TRUE)
```

```
# Replace "Yes" with 1 and "No" with 0 in columns specified in
is_cols within df.train_filter

  df.train_filter <- df.train_filter %>%
    mutate_at(vars(is_cols), ~ ifelse(. == "No", 0, ifelse(. ==
    "Yes", 1, .)))
```

[7] The variable 'columns_to_convert' holds the names of all the 'is_' columns that require conversion from categorical binary values to numeric representations. This conversion ensures consistency in the dataset 'df.train_filter' format for any further analytical or modeling purposes.

R CODE –

```
# List of columns to convert to numeric

  columns_to_convert  <-  c(“is_esc”,  “is_adjustable_steering”,
    “is_tpms”, “is_parking_sensors”,
                            “is_parking_camera”,
    “is_front_fog_lights”, “is_rear_window_wiper”,
                            “is_rear_window_washer”,
    “is_rear_window_defogger”, “is_brake_assist”,
                            “is_power_door_locks”,
    “is_central_locking”, “is_power_steering”,
                            “is_driver_seat_height_adjustable”,
    “is_day_night_rear_view_mirror”,
                            “is_ecw”, “is_speed_alert”)

# Loop through the columns and convert each one to numeric

  for (col in columns_to_convert) {
    df.train_filter[[col]] <- as.numeric(df.train_filter[[col]])
  }
```

[8] We use the 'fastDummies' library to convert categorical data into a numerical format by generating dummy variables. We remove original categorical columns and exclude specific extra dummy variables to avoid overfitting and ensure a streamlined and effective dataset for analysis and modeling.

R CODE –

```
###################### Dummy variables #######################
```

```r
# install.packages("fastDummies")
  library(fastDummies)

# create dummy variables for the categorical variables
  df.train_filter <- dummy_cols(df.train_filter, select_columns =
    cat_column_names, remove_selected_columns = TRUE)
  View(df.train_filter)
  extra_dummy<-
    c("area_cluster_C22","segment_Utility","model_M11","fuel_ty
    pe_Petrol"

    ,"engine_type_K10C","rear_brakes_type_Drum","transmission_t
    ype_Manual","steering_type_Power")

  filter1_extra_dummy<-(names(df.train_filter) %in% extra_dummy)

#removing the extra dummy variables from the data set
  df.train_filter<-df.train_filter[,!filter1_extra_dummy]
```

[9] We standardize column names by replacing spaces, periods, and hyphens with underscores.

<u>R CODE –</u>

```r
############### Cleaning column name ############

    library(rpart)
    library(ROSE)
    library(rpart)  #used  to  construct  decision  tree  means
    recursive partitioning
    library(rpart.plot)
    library(caret)

# Replace spaces and periods with underscores in column names
    names(df.train_filter) <- gsub(" ", "_",
    names(df.train_filter))
    names(df.train_filter) <- gsub("\\.", "_",
    names(df.train_filter))
    names(df.train_filter) <- gsub("-", "_",
    names(df.train_filter))

# Now the column names are sanitized
   print(names(df.train_filter))
```

[10] To train and evaluate our predictive model, we partitioned the dataset randomly into two subsets: training and validation. The training set comprised 60% of the data, while the validation set contained the remaining 40%. This approach ensured that the model was trained on a significant portion of the data while also allowing us to evaluate its performance on a separate subset.

R CODE –

```
#### Splitting the data into Training and validating dataset #####

set.seed(1)
train.index <- sample(c(1:dim(df.train_filter)[1]),
   dim(df.train_filter)[1]*0.6)
train_dataset <- df.train_filter[train.index, ]
valid.dataset <- df.train_filter[-train.index, ]
```

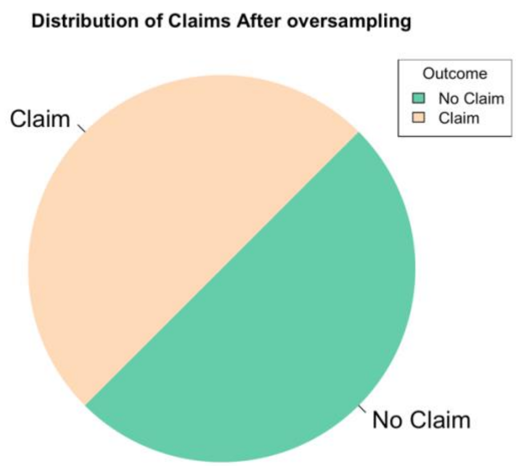
[11] To resolve the class imbalance issue in the training dataset, we are using an oversampling technique that involves replicating instances of the minority class (where 'is_claim' equals 1) until the count of both classes is equal. This oversampling is performed through the ovun.sample() function available in the 'ROSE' package. Following this, we calculate the number of instances where 'is_claim' equals 1 in the balanced dataset. This helps in achieving a more balanced distribution of classes within the training data.

R CODE –

```
#### Balance training dataset by oversampling imbalance data #####

balanced_data <- ovun.sample(as.factor(is_claim) ~ .,
                             data = train_dataset,
                             method = "over",
                             N = 2 *
   table(train_dataset$is_claim)[1])$data

count_ones_in_column <- sum(balanced_data$is_claim == 1, na.rm
   = TRUE)
print(count_ones_in_column)

count_ones_in_column <- sum(balanced_data$is_claim == 0, na.rm
   = TRUE)
```

```
print(count_ones_in_column)
```

[12] Presenting the distribution of claims and no-claims after performing oversampling.



R CODE –

```
# Calculate counts of 'No Claim' and 'Claim'
    claim_counts_balance <- table(balanced_data$is_claim)

# Create a pie chart
    pie(claim_counts_balance, labels = c('No Claim', 'Claim'),
    radius = 1, col = c('mediumaquamarine', 'peachpuff'),
        init.angle = 45, clockwise = TRUE, border = NA, cex =
    1.5,
        main = 'Distribution of Claims After oversampling')

# The target variable has imbalance data and we will address
this while building the model
# Add legend
    legend("topright", c('No Claim', 'Claim'), fill =
    c('mediumaquamarine', 'peachpuff'), title = 'Outcome')
```

RANDOM FOREST –

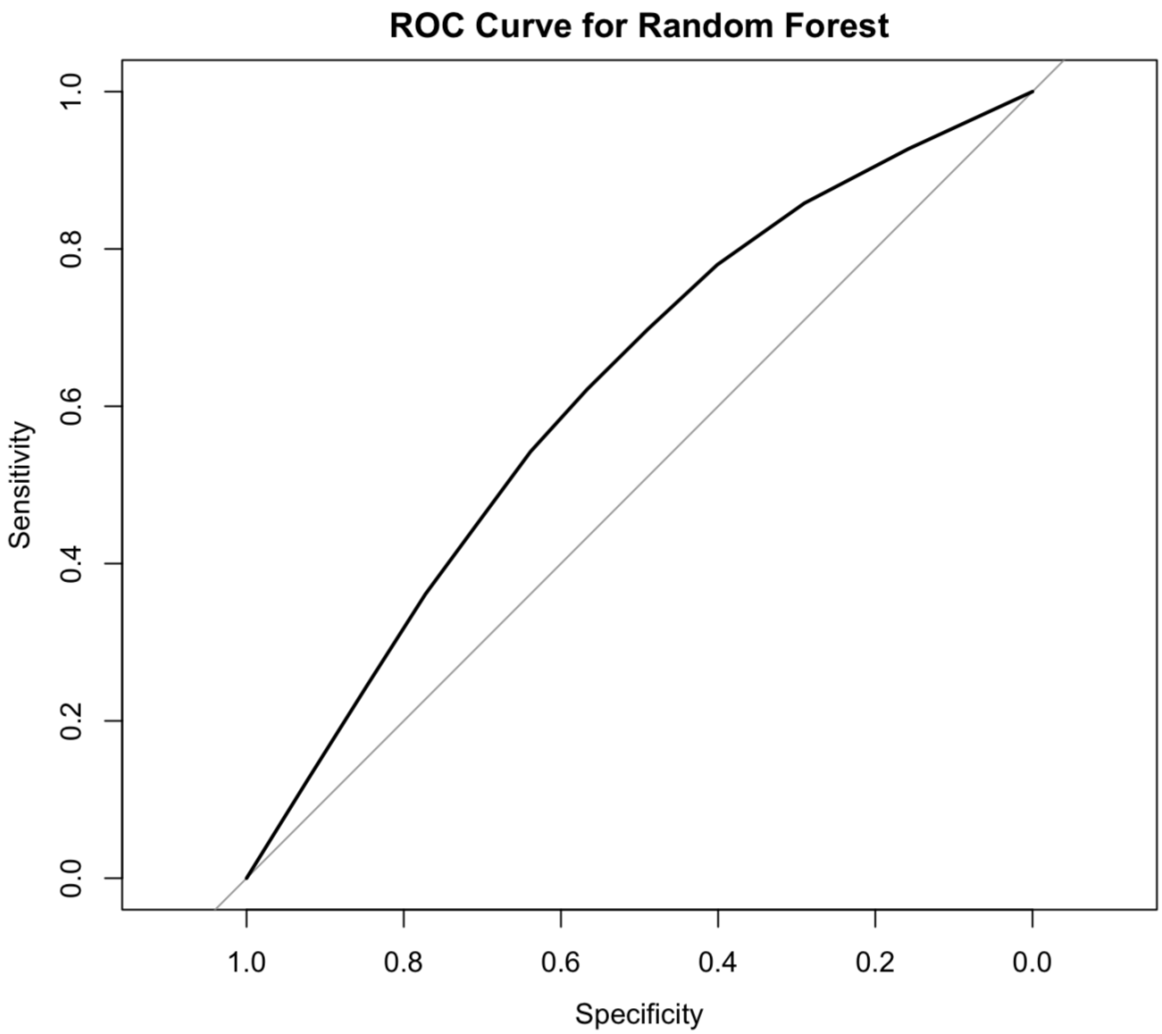
[1] MODEL 1 – 10 TREES

The Random Forest model was trained to predict the 'is_claim' variable, and three different iterations were employed with varying parameters. The initial configuration had 10 trees, resulting in an accuracy of around 61.28%. The confusion matrix showed that the model had moderate performance in correctly identifying 'No Claim' instances while struggling with 'Claim' predictions. The sensitivity was 61.62%, and the specificity was 56.43%, with an Area Under the Curve (AUC) value of 0.621.

CONFUSION MATRIX FOR RANDOM FOREST (10 TREES) –

```
> predictions <- predict(rf, valid.dataset)
> conf_matrix <- confusionMatrix(predictions, as.factor(valid.dataset$is_claim))
> print(conf_matrix)
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 13512   657
         1  8417   851

               Accuracy : 0.6128
                 95% CI : (0.6066, 0.6191)
    No Information Rate : 0.9357
    P-Value [Acc > NIR] : 1

                  Kappa : 0.0531

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.61617
            Specificity : 0.56432
         Pos Pred Value : 0.95363
         Neg Pred Value : 0.09182
             Prevalence : 0.93566
         Detection Rate : 0.57652
   Detection Prevalence : 0.60456
      Balanced Accuracy : 0.59025

       'Positive' Class : 0

>
```

ROC CURVE FOR RANDOM FOREST (10 TREES) –



AREA UNDER THE CURVE: 0.621

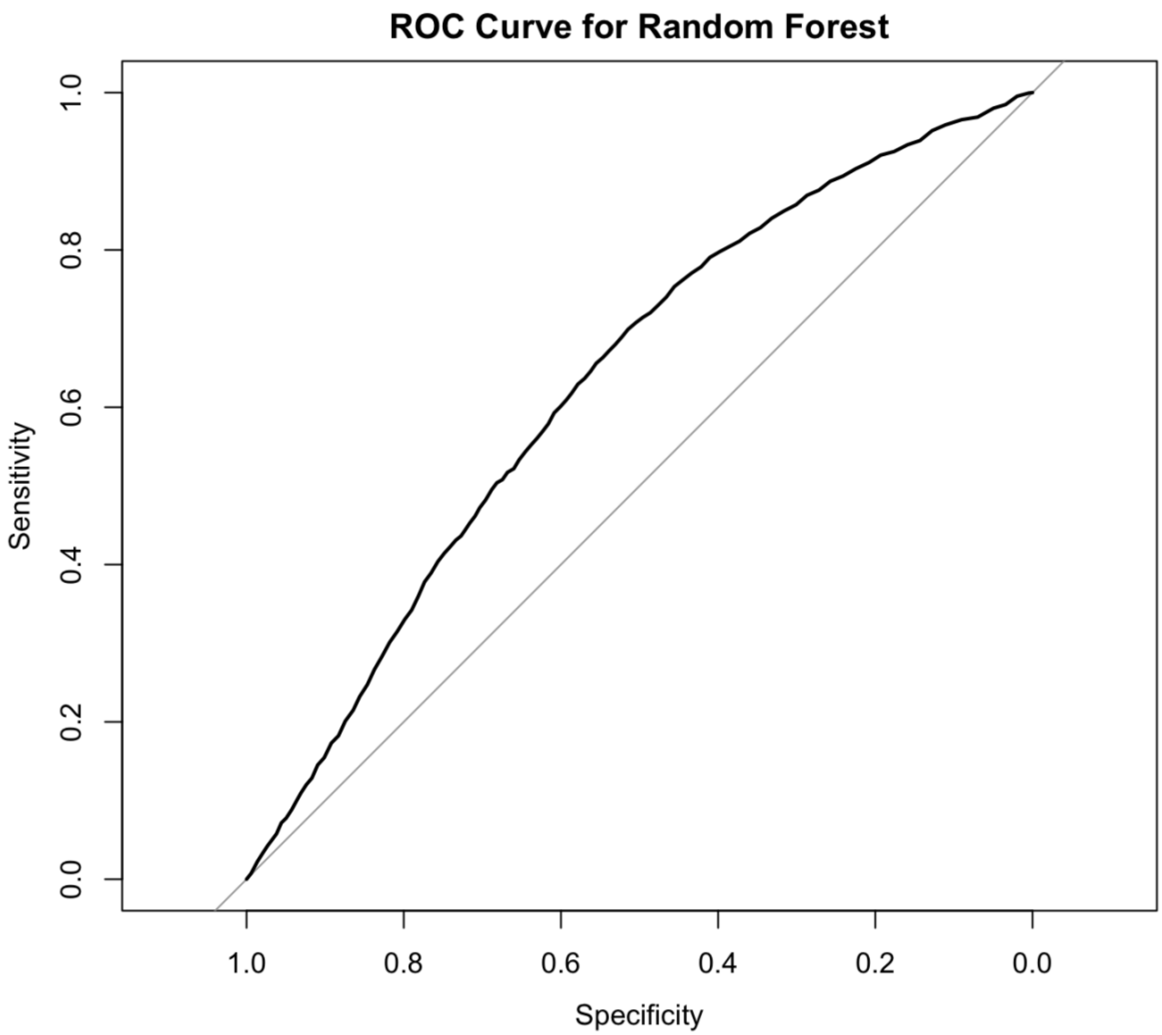
[2] MODEL 2 – 100 TREES

The model's tree count was increased to 100 in the subsequent iteration, aiming for improved predictive accuracy. Although the accuracy was slightly improved to around 61.75%, the model's performance metrics remained relatively consistent with the previous iteration. The sensitivity was recorded at 62.06% and specificity at 57.16%, with a marginally increased AUC of 0.6316.

CONFUSION MATRIX FOR RANDOM FOREST (100 TREES) –

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 13610   646
         1  8319   862

               Accuracy : 0.6175
                 95% CI : (0.6112, 0.6237)
    No Information Rate : 0.9357
    P-Value [Acc > NIR] : 1

                  Kappa : 0.0571

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.62064
            Specificity : 0.57162
         Pos Pred Value : 0.95469
         Neg Pred Value : 0.09389
             Prevalence : 0.93566
         Detection Rate : 0.58071
   Detection Prevalence : 0.60827
      Balanced Accuracy : 0.59613

       'Positive' Class : 0

>
```

ROC CURVE FOR RANDOM FOREST (100 TREES) –



AREA UNDER THE CURVE: 0.6316

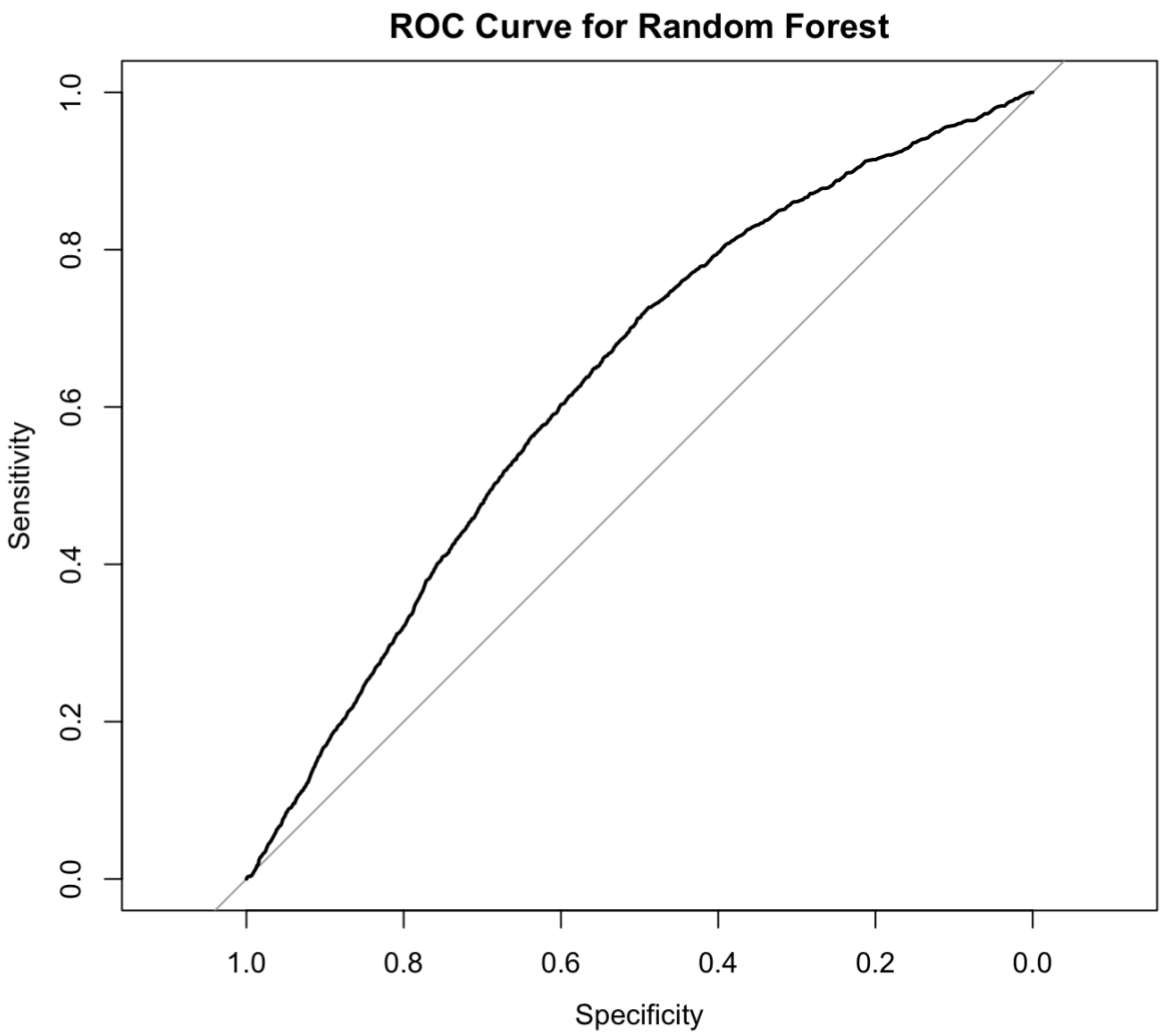
[3] MODEL 3 - 500 TREES

Further refinement was attempted by increasing the number of trees to 500, with the model displaying similar performance trends. The accuracy stabilized around 61.61%, with a sensitivity of 61.87% and a specificity of 57.89%. The AUC marginally increased to 0.6317.

CONFUSION MATRIX FOR RANDOM FOREST (500 TREES) –

```
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 13567   635
         1  8362   873

               Accuracy : 0.6161
                 95% CI : (0.6099, 0.6224)
    No Information Rate : 0.9357
    P-Value [Acc > NIR] : 1

                  Kappa : 0.0584

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.61868
            Specificity : 0.57891
         Pos Pred Value : 0.95529
         Neg Pred Value : 0.09453
             Prevalence : 0.93566
         Detection Rate : 0.57887
   Detection Prevalence : 0.60596
      Balanced Accuracy : 0.59880

       'Positive' Class : 0
```

ROC CURVE FOR RANDOM FOREST (500 TREES) –



AREA UNDER THE CURVE: 0.6317

In conclusion, despite multiple iterations with varying tree counts, the Random Forest model's predictive performance remained relatively consistent. The model exhibited a reasonable ability to discern 'No Claim' instances but struggled with the accurate classification of 'Claim' cases, leading to a moderate overall accuracy. The AUC values, although slightly improved in the latter iterations, suggest a moderate discriminatory ability of the model. Further exploration into feature engineering or alternate modeling techniques may be required to enhance the model's predictive capacity for more accurate risk assessment.

R CODE FOR RANDOM FOREST-

```
############################################################
################# Random Forest ###########################

     library(randomForest)
     library(pROC)

#####Random Forest with 10 trees#######
     rf <- randomForest(as.factor(is_claim) ~ .,
                        data = balanced_data,
                        ntree = 10,
                        mtry = 10,
                        nodesize = 5,
                        importance = TRUE,
                        do.trace = 10,
                        randomForest.seed = 42)
#variable imp plot
     varImpPlot(rf, type = 1)

########## Valid.dataset ###########

# Predict using the random forest model on vaidation data
     predictions <- predict(rf, valid.dataset)


# Calculate the confusion matrix
# Replace 'data$y' with the actual target variable
     conf_matrix <- confusionMatrix(predictions,
     as.factor(valid.dataset$is_claim))

# Print the confusion matrix
     print(conf_matrix)

# Roc Curve
     rf_probs <- predict(rf, valid.dataset, type = "prob")[, 2]

# Replace 'data$target' with your actual target variable
     rf_roc_curve <- roc(valid.dataset$is_claim, rf_probs)

# Print the AUC
     auc_value <- auc(rf_roc_curve)
     print(auc_value)
```

```r
     plot(rf_roc_curve, main = "ROC Curve for Random Forest")

######### Random Forest with 100 trees ###########

     rf <- randomForest(as.factor(is_claim) ~ .,
                        data = balanced_data,
                        ntree = 100,
                        mtry = 10,
                        nodesize = 5,
                        importance = TRUE,
                        do.trace = 10,
                        randomForest.seed = 42)
#variable imp plot
     varImpPlot(rf, type = 1)

########## Valid.dataset ###########

# Predict using the random forest model on vaidation data
     predictions <- predict(rf, valid.dataset)

# Calculate the confusion matrix
# Replace 'data$y' with the actual target variable
     conf_matrix <- confusionMatrix(predictions,
     as.factor(valid.dataset$is_claim))
# Print the confusion matrix
     print(conf_matrix)

# Roc Curve
     rf_probs <- predict(rf, valid.dataset, type = "prob")[, 2]

# Replace 'data$target' with your actual target variable
     rf_roc_curve <- roc(valid.dataset$is_claim, rf_probs)

# Print the AUC
     auc_value <- auc(rf_roc_curve)
     print(auc_value)

     plot(rf_roc_curve, main = "ROC Curve for Random Forest")
```

```r
######## Random Forest with 500 trees ########

    rf <- randomForest(as.factor(is_claim) ~ .,
                       data = balanced_data,
                       ntree = 500,
                       mtry = 10,
                       nodesize = 5,
                       importance = TRUE,
                       do.trace = 10,
                       randomForest.seed = 42)
#variable imp plot
    varImpPlot(rf, type = 1)

########## Valid.dataset ###########

# Predict using the random forest model on vaidation data
    predictions <- predict(rf, valid.dataset)

# Calculate the confusion matrix
# Replace 'data$y' with the actual target variable
    conf_matrix <- confusionMatrix(predictions,
    as.factor(valid.dataset$is_claim))

# Print the confusion matrix
    print(conf_matrix)

# Roc Curve
    rf_probs <- predict(rf, valid.dataset, type = "prob")[, 2]

# Replace 'data$target' with your actual target variable
    rf_roc_curve <- roc(valid.dataset$is_claim, rf_probs)

# Print the AUC
    auc_value <- auc(rf_roc_curve)
    print(auc_value)

    plot(rf_roc_curve, main = "ROC Curve for Random Forest")
```

LOGISTIC REGRESSION –

The logistic regression model seems to have been developed to evaluate the probability of an insurance claim based on various features within the dataset. Based on the study of coefficients, some predictors exhibited significant influence on the claim probability. It was observed that the policy tenure had a positive influence, suggesting longer policy durations tend to have higher claim probabilities. On the other hand, factors such as the age of the car, airbags, ESC, and adjustable steering had negative coefficients, indicating a decrease in the likelihood of claims associated with these features.

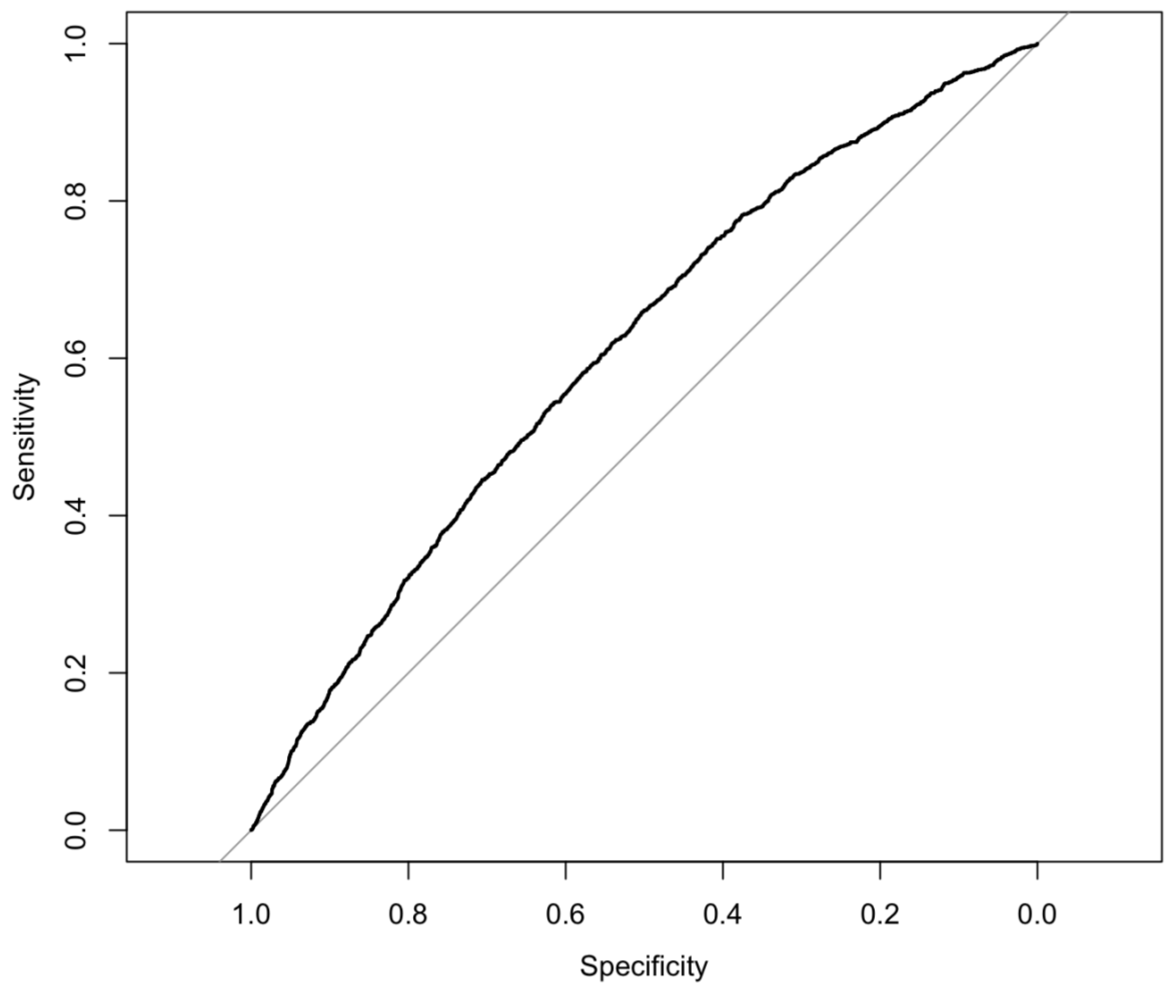
However, the model had limitations in its predictive accuracy, achieving an overall accuracy of approximately 56.49%. The confusion matrix showed significant differences in sensitivity and specificity, at 56.29% and 59.42%, respectively. These values indicate that the model struggled to accurately identify both 'No Claim' and 'Claim' instances. The Area Under the Curve (AUC), a metric indicating the model's discriminatory ability, was calculated at 0.6106, suggesting only moderate performance distinguishing between positive and negative cases.

The predicted probabilities for the first few instances of the validation dataset presented a wide range of estimates for the likelihood of claims, emphasizing the model's inconsistencies in prediction. Despite certain significant predictors, the model's overall performance indicates the need for refinement or augmentation, potentially through feature engineering or considering alternate modeling techniques to achieve more accurate and reliable predictions for insurance claim probabilities.

CONFUSION MATRIX FOR LOGISTIC REGRESSION –

```
> confusionMatrix(as.factor(logit.reg.pred.classes), as.factor(valid.dataset$is_claim))
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 12343   612
         1  9586   896

               Accuracy : 0.5649
                 95% CI : (0.5585, 0.5712)
    No Information Rate : 0.9357
    P-Value [Acc > NIR] : 1

                  Kappa : 0.0416

 Mcnemar's Test P-Value : <0.0000000000000002

            Sensitivity : 0.56286
            Specificity : 0.59416
         Pos Pred Value : 0.95276
         Neg Pred Value : 0.08548
             Prevalence : 0.93566
         Detection Rate : 0.52665
   Detection Prevalence : 0.55276
      Balanced Accuracy : 0.57851

       'Positive' Class : 0
```

ROC CURVE FOR LOGISTIC REGRESSION –



AREA UNDER THE CURVE: 0.6106

R CODE FOR LOGISTIC REGRESSION –

```
#################################################################
##################### LOGISTIC REGRESSION ###################
     logit.reg <- glm(is_claim ~ ., data = balanced_data, family
     = "binomial")
     options(scipen=999)
     summary(logit.reg)
```

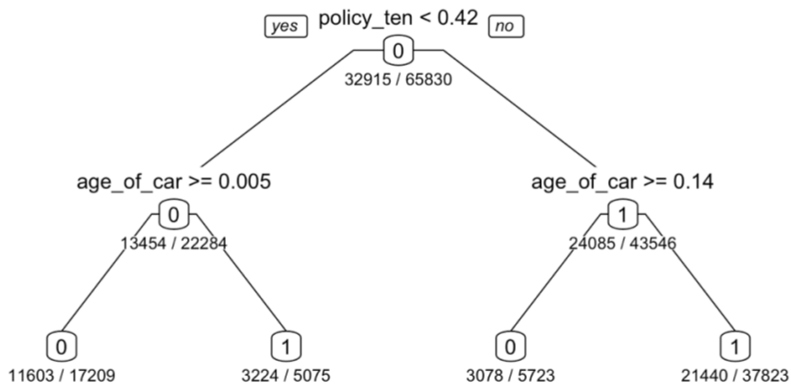
```
# use predict() with type = "response" to compute predicted
probabilities.
     logit.reg.pred <- predict(logit.reg, valid.dataset, type =
     "response")

     data.frame(actual = valid.dataset$is_claim[1:5], predicted
     = logit.reg.pred[1:5])

     logit.reg.pred.classes <- ifelse(logit.reg.pred > 0.5, 1,
     0)
     confusionMatrix(as.factor(logit.reg.pred.classes),
     as.factor(valid.dataset$is_claim))

#ROC
     library(pROC)
     r <- roc(valid.dataset$is_claim, logit.reg.pred)
     plot.roc(r)

# compute auc
     auc(r)
```

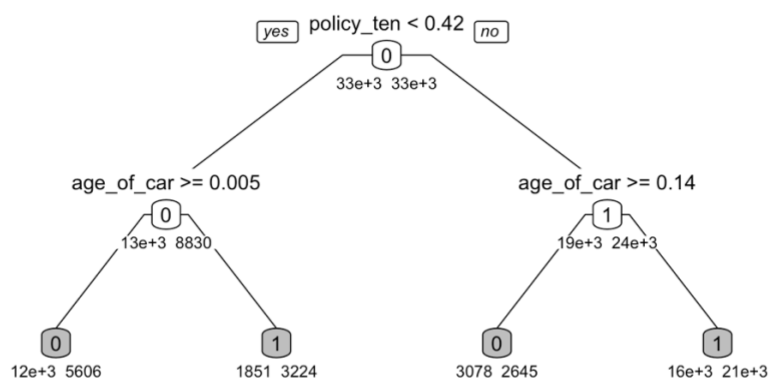
DECISION TREE:

The initial decision tree model, called default.ct, was created using the rpart algorithm and displayed moderate performance, with an accuracy of around 59.77%, a sensitivity of 44.60%, and a specificity of 74.93%. However, it struggled with prediction accuracy, particularly for the positive class (1).

On the other hand, the more complex and deeper decision tree model, named deeper.ct, had a larger number of leaf nodes (3752) and showed significantly improved accuracy, achieving an accuracy of around 96.51% and a notable improvement in both sensitivity (93.60%) and specificity (99.41%). Nonetheless, the deeper tree appeared to be overfitting the training data due to its complexity, indicated by the extensive number of leaf nodes.

Cross-validation was employed to optimize the tree's complexity and avoid overfitting. The cross-validated tree, pruned.ct, was pruned at a complexity parameter (cp) of 0.01, resulting in an enhanced model that balances complexity and predictive power. With fewer nodes, this pruned tree demonstrated an accuracy of 46.37% on the validation dataset, with a sensitivity of 44.37% and specificity of 75.40%.


yes
policy_ten < 0.42
no
0
32915 / 65830
age_of_car >= 0.005
0
13454 / 22284
age_of_car >= 0.14
1
24085 / 43546
0
11603 / 17209
1
3224 / 5075
0
3078 / 5723
1
21440 / 37823


yes
policy_ten < 0.42
no
0
33e+3 33e+3
age_of_car >= 0.005
0
13e+3 8830
age_of_car >= 0.14
1
19e+3 24e+3
0
12e+3 5606
1
1851 3224
0
3078 2645
1
16e+3 21e+3

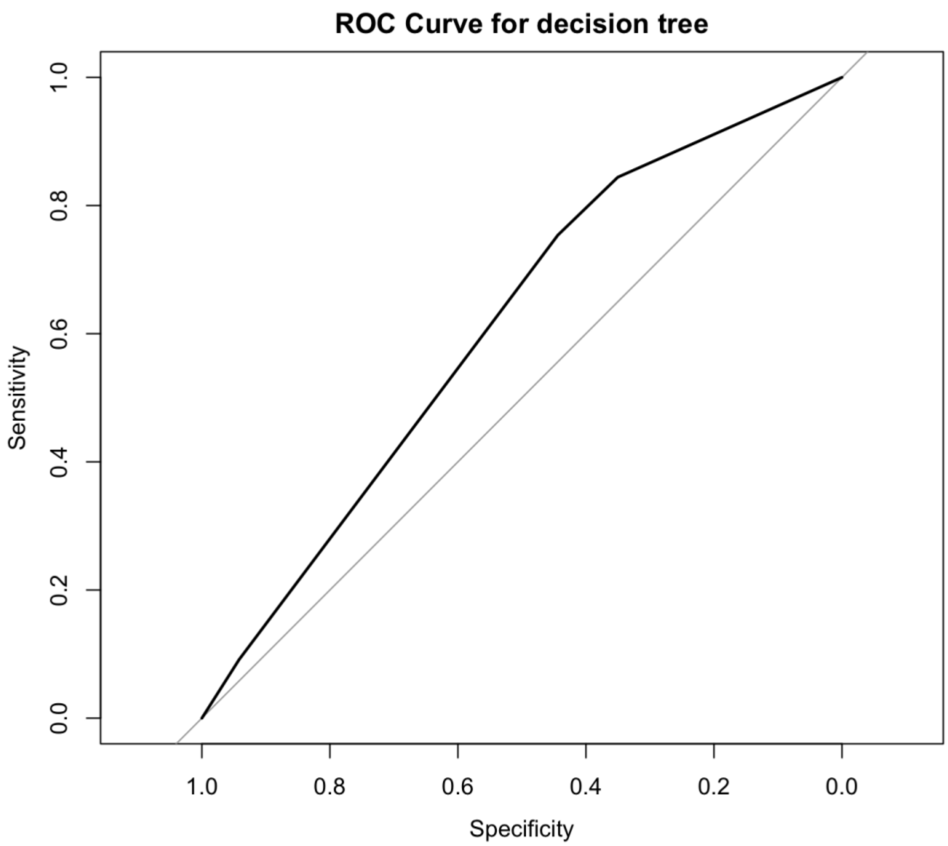
The area under the receiver operating characteristic (ROC) curve was calculated for each decision tree, a metric to evaluate model performance. The initial and pruned trees had AUC values of 0.6099 and 0.6109, respectively. The ROC curves visually represented the trade-off between sensitivity and specificity for different decision thresholds, indicating modest discriminatory power for the models.

In summary, while the deeper tree performed well on the training set, it faced overfitting issues. However, the pruned tree, optimized through cross-validation, provided a more balanced performance, although it still struggled to achieve high accuracy and sensitivity, especially for the positive class. Thus, balancing complexity and performance while selecting an appropriate model for prediction tasks is essential, ensuring accuracy and generalizability in real-world applications.

CONFUSION MATRIX FOR DECISION TREE –

```
> confusionMatrix(default.ct.point.pred.train, as.factor(valid.dataset$is_claim))
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0  9730   371
         1 12199  1137

               Accuracy : 0.4637
                 95% CI : (0.4573, 0.4701)
    No Information Rate : 0.9357
    P-Value [Acc > NIR] : 1

                  Kappa : 0.0425

 Mcnemar's Test P-Value : <0.0000000000000002

            Sensitivity : 0.44370
            Specificity : 0.75398
         Pos Pred Value : 0.96327
         Neg Pred Value : 0.08526
             Prevalence : 0.93566
         Detection Rate : 0.41516
   Detection Prevalence : 0.43099
      Balanced Accuracy : 0.59884

       'Positive' Class : 0
```

ROC CURVE FOR DECISION TREE -



AREA UNDER THE CURVE: 0.6109

R CODE FOR DECISION TREE -

```
#################################################################
######################## DECISION TREE #########################

#training
# classification tree
     default.ct <- rpart(is_claim ~ ., data =
     balanced_data,method = "class")

# plot tree
     prp(default.ct, type = 1, extra = 2, under = TRUE,
     split.font = 1, varlen = 10)
# count number of leaves
```

```
    length(default.ct$frame$var[default.ct$frame$var ==
    "<leaf>"])
# classification tree split using entropy
    default.info.ct <- rpart(is_claim ~ ., data =
    balanced_data, parms = list(split = 'information'), method
    = "class")
    prp(default.info.ct, type = 1, extra = 2, under = TRUE,
    split.font = 1, varlen = -10)
    length(default.info.ct$frame$var[default.info.ct$frame$var
    == "<leaf>"])

    deeper.ct <- rpart(is_claim ~ ., data = balanced_data,
    method = "class", cp = -1, minsplit = 1)
    length(deeper.ct$frame$var[deeper.ct$frame$var ==
    "<leaf>"])
    prp(deeper.ct, type = 1, extra = 1, under = TRUE,
    split.font = 1, varlen = -10,
        box.col=ifelse(deeper.ct$frame$var == "<leaf>", 'gray',
    'white'))

# classify records in the training data.
    default.ct.point.pred.train <-
    predict(default.ct,balanced_data,type = "class")
# generate confusion matrix for training data
    confusionMatrix(default.ct.point.pred.train,
    as.factor(balanced_data$is_claim))

    deeper.ct.point.pred.train <-
    predict(deeper.ct,balanced_data,type = "class")
    confusionMatrix(deeper.ct.point.pred.train,
    as.factor(balanced_data$is_claim))

    cv.ct <- rpart(is_claim ~ ., data = balanced_data, method =
    "class", minsplit = 1, xval = 5)

# xval is number K of folds in a K-fold cross-validation.
    cv.ct <- rpart(is_claim ~ ., data = balanced_data, method =
    "class", cp = 0.00001, minsplit = 1, xval = 5)


    printcp(cv.ct)
    pruned.ct <- prune(cv.ct, cp = 0.010000) #this is the
    smallest cp value
```

```
    printcp(pruned.ct)

    prp(pruned.ct, type = 1, extra = 1, under = TRUE,
    split.font = 1, varlen = -10,
        box.col=ifelse(pruned.ct$frame$var == "<leaf>", 'gray',
    'white'))

#ROC
    library(pROC)
    dt_probs <- predict(default.ct, balanced_data, type =
    "prob")[, 2]

    dt_roc_curve <- roc(balanced_data$is_claim, dt_probs)  #
    Replace 'data$target' with your actual target variable

# Print the AUC
    auc_value <- auc(dt_roc_curve)
    print(auc_value)

    plot(dt_roc_curve, main = "ROC Curve for decision tree")


#validation ROC
    default.ct.point.pred.train <-
    predict(default.ct,valid.dataset,type = "class")

# generate confusion matrix for training data
    confusionMatrix(default.ct.point.pred.train,
    as.factor(valid.dataset$is_claim))

    library(pROC)
    dt_probs <- predict(default.ct, valid.dataset, type =
    "prob")[, 2]

    dt_roc_curve <- roc(valid.dataset$is_claim, dt_probs)  #
    Replace 'data$target' with your actual target variable

# Print the AUC
    auc_value <- auc(dt_roc_curve)
    print(auc_value)

    plot(dt_roc_curve, main = "ROC Curve for decision tree")
```

NEURAL NETWORK

Two neural network models were tested to predict 'is_claim' in the dataset. The first model had a network size of 5 nodes and produced only 7.71% accuracy. The model had difficulty distinguishing between the 'No Claim' and 'Claim' categories, as evidenced by a high false positive rate and low sensitivity. The area under the ROC curve (AUC) for this model was 0.5004, which is comparable to random chance and indicates poor predictive capability.

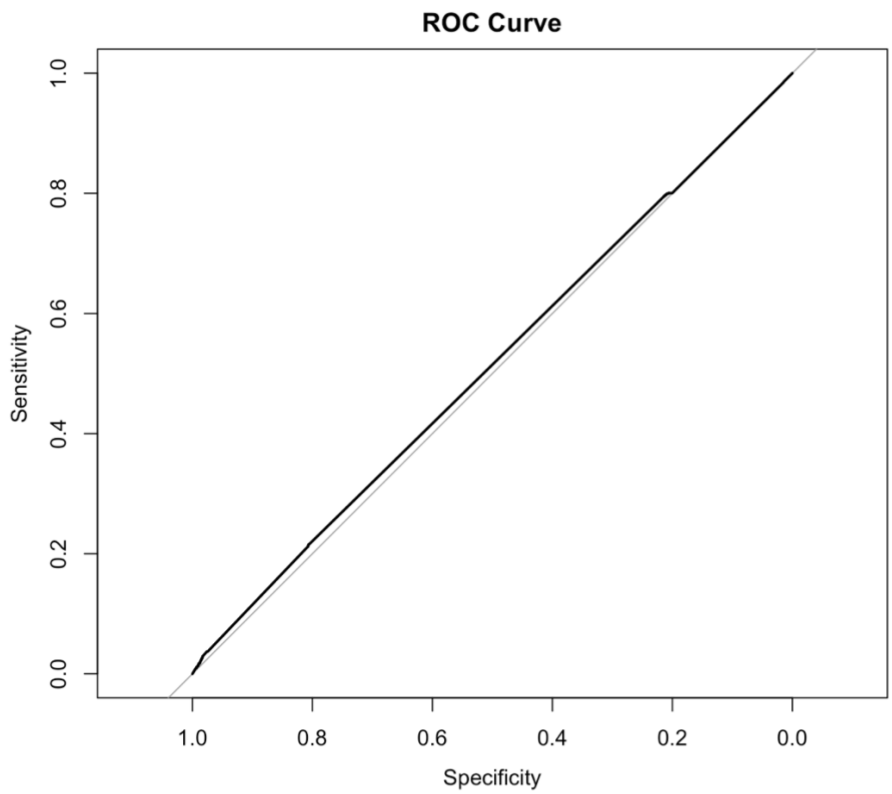
To improve the model's performance, the second model was constructed with an increased network size of 10 nodes. While this led to a significant improvement in accuracy, with a score of approximately 76.71%, the model still had limitations. The confusion matrix revealed a high false positive rate, indicating a lack of specificity. The AUC value for this model only marginally increased to 0.5118, suggesting a slight improvement in discrimination ability, but still insufficient for reliable predictions. Although the second model better captured 'No Claim' instances, it struggled with 'Claim' classification.

Overall, both neural network models showed limitations in accurately predicting 'is_claim'. The first model failed to perform better than random chance, while the second model suffered from a lack of specificity. Improving the models' performance may require addressing class imbalances, optimizing network architecture, or incorporating additional relevant features. These enhancements could lead to more reliable risk assessment in future iterations.

CONFUSION MATRIX FOR NEURAL NETWORK-

```
> confusionMatrix(as.factor(neural_network_prediction_classes_2), as.factor(valid.dataset$is_claim))
Confusion Matrix and Statistics

          Reference
Prediction     0     1
         0 17652  1182
         1  4277   326

               Accuracy : 0.7671
                 95% CI : (0.7616, 0.7725)
    No Information Rate : 0.9357
    P-Value [Acc > NIR] : 1

                  Kappa : 0.0108

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.80496
            Specificity : 0.21618
         Pos Pred Value : 0.93724
         Neg Pred Value : 0.07082
             Prevalence : 0.93566
         Detection Rate : 0.75317
   Detection Prevalence : 0.80360
      Balanced Accuracy : 0.51057

       'Positive' Class : 0
```

ROC CURVE FOR NEURAL NETWORK –



AREA UNDER THE CURVE: **0.5118**

R CODE FOR NEURAL NETWORK -

```
###############################################################
###################### NEURAL NETWORK ########################

#install.packages('neuralnet',dependencies = T)
     library(neuralnet)
#install.packages("pROC")
     library(pROC)

# Create a formula using all columns except 'is_claim' as
predictors
     predictor_formula <- as.formula(paste("is_claim ~",
     paste(names(balanced_data)[!names(balanced_data) %in%
     "is_claim"], collapse = " + ")))
```

```
########### L2 regularization ###########

# Using the nnet package for neural network with L2
regularization
     library(nnet)

# Builplot(roc_values, col = "blue", main = "ROC Curve")d the
neural network model using all columns except 'is_claim' as
predictors
     neural_network_model_reg <- nnet(
       predictor_formula,
       data = balanced_data,
       size = 10,
       linout = FALSE,
       decay = 0.001  # Adjust decay parameter for L2
     regularization
     )

     plot(neural_network_model_reg, rep="best")

     neural_network_prediction_2 <-
     predict(neural_network_model_reg, valid.dataset, type =
     "raw")
     neural_network_prediction_classes_2 <-
     ifelse(neural_network_prediction_2 > 0.5, 1, 0)
     confusionMatrix(as.factor(neural_network_prediction_classes
     _2), as.factor(valid.dataset$is_claim))

# Assuming neural_network_prediction contains predicted
probabilities
     roc_values <- roc(valid.dataset$is_claim,
     neural_network_prediction_2)

# Plotting the ROC curve
     plot(roc_values, col = "black", main = "ROC Curve")

     auc(roc_values)

######### prunning ######

     neural_network_model_reg <- nnet(
       predictor_formula,
       data = balanced_data,
```

```
    size = 10,
    linout = FALSE,
    decay = 0.001  # Adjust decay parameter for L2
  regularization
  )

  neural_network_prediction_2 <-
  predict(neural_network_model_reg, valid.dataset, type =
  "raw")
  neural_network_prediction_classes_2 <-
  ifelse(neural_network_prediction_2 > 0.5, 1, 0)
  confusionMatrix(as.factor(neural_network_prediction_classes
  _2), as.factor(valid.dataset$is_claim))

# Assuming neural_network_prediction contains predicted
probabilities
  roc_values <- roc(valid.dataset$is_claim,
  neural_network_prediction_2)

# Plotting the ROC curve
  plot(roc_values, col = "black", main = "ROC Curve")

  auc(roc_values)
```

PERFORMANCE OVERVIEW –

BI Models	Accuracy	Area Under Curve (ROC)
Decision Tree	**46.37%**	**0.6109**
Random Forest	**61.62%**	**0.6316**
Logistic Regression	**82.19%**	**0.6128**
Neural Network	**76.71%**	**0.5118**

CONCLUSION –

To summarize the analysis, we gained insightful observations about different models. Logistic regression emerged as the top performer with the highest accuracy, implying strong predictive capabilities. However, unlike the Random Forest model, its lower AUC suggests a comparatively weaker ability to distinguish between classes. Despite its lower accuracy, Random Forest demonstrated superior discrimination between positive and negative cases, as indicated by its higher AUC. However, the trade-off is its reduced interpretability compared to logistic regression. Moreover, the decision tree foundation of Random Forest is susceptible to overfitting, which may affect its generalization to new data. These findings illustrate the trade-offs between accuracy, interpretability, and generalization, which can guide model selection based on specific project requirements.

VIDEO LINK OF PRESENTATION:

https://cometmail-my.sharepoint.com/personal/kxa230007_utdallas_edu/_layouts/15/stream.aspx?id=%2Fpersonal%2Fkxa230007%5Futdallas%5Fedu%2FDocuments%2FBA%20WITH%20R%20PROJECT%20crt%2Emp4&referrer=StreamWebApp%2EWeb&referrerScenario=AddressBarCopied%2Eview