###Risk Assessment Model for Diabetic Patient Readmission

##Gowtham Venkata Sai Ram Maddala ##Vishnu Sangadala ##Ragu Ram Kotaru ##Raghu Varma Kosuri ##Venkata Shanmukh Reddy Atluru

```
set.seed(123)
```

Three Research Questions and Their Implementations ###1) How does handling class imbalance and missing values impact model performance in predicting hospital readmissions? ###2) Can logistic regression, a generalized linear model (GLM), provide reliable predictions for hospital readmission outcomes? ###3) Do more complex machine learning models, such as decision trees and random forests, outperform simpler models in predicting hospital readmissions?

```
# Loading libraries
library(tidyverse)   # Includes dplyr, ggplot2, etc.
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.4     v tidyr     1.3.1
## v purrr     1.0.4
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```
library(data.table)  # For fast data manipulation
```

```
##
## Attaching package: 'data.table'
##
## The following objects are masked from 'package:lubridate':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year
##
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
##
## The following object is masked from 'package:purrr':
##
##     transpose
```

```
library(ggplot2)     # For plotting (similar to matplotlib)
library(reshape2)    # Helpful for reshaping data
```

```
##
## Attaching package: 'reshape2'
##
```

```
## The following objects are masked from 'package:data.table':
##
##     dcast, melt
##
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
library(cowplot) # Enhances ggplot2 visualizations
```

```
##
## Attaching package: 'cowplot'
##
## The following object is masked from 'package:lubridate':
##
##     stamp
```

```
library(dplyr)
```

##dataset reading and missing values

```
df <- read.csv("diabetic_data.csv", stringsAsFactors = FALSE)
# Loop through columns and count '?' in character columns
for (col in names(df)) {
  if (is.character(df[[col]])) {
    count_qmark <- sum(df[[col]] == '?')
    if (count_qmark > 0) {
      cat(col, count_qmark, "\n")
    }
  }
}
```

```
## race 2273
## weight 98569
## payer_code 40256
## medical_specialty 49949
## diag_1 21
## diag_2 358
## diag_3 1423
```

##gender missing values and unknown

```
# gender was coded differently so we use a custom count for this one
cat("gender", sum(df$gender == "Unknown/Invalid"), "\n")
```

```
## gender 3
```

1)How does handling class imbalance and missing values impact model performance in predicting hospital readmissions?

##handling missing values

```r
# Dropping columns with large number of missing values
df <- dplyr::select(df, -c(weight, payer_code, medical_specialty))

# Creating a set of row indices to drop
drop_Idx <- which(df$diag_1 == '?' & df$diag_2 == '?' & df$diag_3 == '?')

drop_Idx <- union(drop_Idx, which(df$diag_1 == '?'))
drop_Idx <- union(drop_Idx, which(df$diag_2 == '?'))
drop_Idx <- union(drop_Idx, which(df$diag_3 == '?'))
drop_Idx <- union(drop_Idx, which(df$race == '?'))
drop_Idx <- union(drop_Idx, which(df$discharge_disposition_id == 11))
drop_Idx <- union(drop_Idx, which(df$gender == 'Unknown/Invalid'))

# Keeping only the remaining indices
df <- df[-drop_Idx, ]

# Checking for remaining '?' values in character columns
for (col in colnames(df)) {
    if (is.character(df[[col]]) || is.factor(df[[col]])) {
        count <- sum(df[[col]] == "?", na.rm = TRUE)
        cat(col, count, "\n")
    }
}
```

```
## race 0
## gender 0
## age 0
## diag_1 0
## diag_2 0
## diag_3 0
## max_glu_serum 0
## A1Cresult 0
## metformin 0
## repaglinide 0
## nateglinide 0
## chlorpropamide 0
## glimepiride 0
## acetohexamide 0
## glipizide 0
## glyburide 0
## tolbutamide 0
## pioglitazone 0
## rosiglitazone 0
## acarbose 0
## miglitol 0
## troglitazone 0
## tolazamide 0
## examide 0
## citoglipton 0
## insulin 0
## glyburide.metformin 0
## glipizide.metformin 0
## glimepiride.pioglitazone 0
```

```
## metformin.rosiglitazone 0
## metformin.pioglitazone 0
## change 0
## diabetesMed 0
## readmitted 0
```

##these columns have same value so dropping these as these do not provide any information or add value

```
df <- df[, !names(df) %in% c("citoglipton", "examide")]
```

```
colnames(df)
```

```
##  [1] "encounter_id"             "patient_nbr"
##  [3] "race"                     "gender"
##  [5] "age"                      "admission_type_id"
##  [7] "discharge_disposition_id" "admission_source_id"
##  [9] "time_in_hospital"         "num_lab_procedures"
## [11] "num_procedures"           "num_medications"
## [13] "number_outpatient"        "number_emergency"
## [15] "number_inpatient"         "diag_1"
## [17] "diag_2"                   "diag_3"
## [19] "number_diagnoses"         "max_glu_serum"
## [21] "A1Cresult"                "metformin"
## [23] "repaglinide"              "nateglinide"
## [25] "chlorpropamide"           "glimepiride"
## [27] "acetohexamide"            "glipizide"
## [29] "glyburide"                "tolbutamide"
## [31] "pioglitazone"             "rosiglitazone"
## [33] "acarbose"                 "miglitol"
## [35] "troglitazone"             "tolazamide"
## [37] "insulin"                  "glyburide.metformin"
## [39] "glipizide.metformin"      "glimepiride.pioglitazone"
## [41] "metformin.rosiglitazone"  "metformin.pioglitazone"
## [43] "change"                   "diabetesMed"
## [45] "readmitted"
```

```
df1 <- df
```

##feature engineering

```
# Step 1: Create service_utilization column
df$service_utilization <- df$number_outpatient + df$number_emergency + df$number_inpatient
# List of medication columns
keys <- c('metformin', 'repaglinide', 'nateglinide', 'chlorpropamide', 'glimepiride',
          'glipizide', 'glyburide', 'pioglitazone', 'rosiglitazone', 'acarbose',
          'miglitol', 'insulin', 'glyburide.metformin', 'tolazamide',
          'metformin.pioglitazone','metformin.rosiglitazone',
          'glimepiride.pioglitazone', 'glipizide.metformin', 'troglitazone',
          'tolbutamide', 'acetohexamide')

# Convert relevant columns to character
df[keys] <- lapply(df[keys], as.character)
```

```r
# Create a temp binary matrix: 1 if changed (i.e., not 'No' or 'Steady'), 0 otherwise
med_change_matrix <- sapply(df[keys], function(x) ifelse(x == "No" | x == "Steady", 0, 1))

# Sum across rows to get number of changes
df$numchange <- rowSums(med_change_matrix, na.rm = TRUE)

# Count the number of patients by number of changes
table(df$numchange)
```

```
##
##     0     1     2     3     4
## 70142 24922  1271   106     5
```

##feature engineering - 2 # Created a new feature representing the total number of medication changes during the patient's hospital stay. # The dataset includes 23 features indicating whether each of 23 diabetes-related medications (or combinations) was changed. # Instead of tracking individual drug changes, we summed them to simplify the model and capture any general trend # between the number of medication changes and readmission likelihood, as prior research suggests such changes may reduce readmissions.

```r
# Re-encoding 'admission_type_id'
df$admission_type_id[df$admission_type_id %in% c(2, 7)] <- 1
df$admission_type_id[df$admission_type_id == 6] <- 5
df$admission_type_id[df$admission_type_id == 8] <- 5

# Re-encoding 'discharge_disposition_id'
df$discharge_disposition_id[df$discharge_disposition_id %in% c(6, 8, 9, 13)] <- 1
df$discharge_disposition_id[df$discharge_disposition_id %in% c(3, 4, 5, 14, 22, 23, 24)] <- 2
df$discharge_disposition_id[df$discharge_disposition_id %in% c(12, 15, 16, 17)] <- 10
df$discharge_disposition_id[df$discharge_disposition_id %in% c(25, 26)] <- 18

# Re-encoding 'admission_source_id'
df$admission_source_id[df$admission_source_id %in% c(2, 3)] <- 1
df$admission_source_id[df$admission_source_id %in% c(5, 6, 10, 22, 25)] <- 4
df$admission_source_id[df$admission_source_id %in% c(15, 17, 20, 21)] <- 9
df$admission_source_id[df$admission_source_id %in% c(13, 14)] <- 11
```

Encoded categorical variables such as gender, race, medication change status, and the 23 drug-related features into numeric binary values.

This transformation helps the model interpret the data more effectively.

For instance, the 'medication change' feature was converted from "No" (no change) and "Ch" (changed) to 0 and 1, respectively.

```
# Encoding 'change' variable
df$change[df$change == 'Ch'] <- 1
df$change[df$change == 'No'] <- 0

# Encoding 'gender' variable
df$gender[df$gender == 'Male'] <- 1
df$gender[df$gender == 'Female'] <- 0

# Encoding 'diabetesMed' variable
df$diabetesMed[df$diabetesMed == 'Yes'] <- 1
df$diabetesMed[df$diabetesMed == 'No'] <- 0

# Loop through each medication column and replace values
for (col in keys) {
  df[[col]] <- ifelse(df[[col]] == 'No', 0,
                      ifelse(df[[col]] == 'Steady', 1,
                             ifelse(df[[col]] == 'Up', 1, 1)))  # Replace 'Up' and 'Down' as 1
}
```

```
# Replacing values in 'A1Cresult' column
df$A1Cresult[df$A1Cresult == '>7'] <- 1
df$A1Cresult[df$A1Cresult == '>8'] <- 1
df$A1Cresult[df$A1Cresult == 'Norm'] <- 0
df$A1Cresult[df$A1Cresult == 'None'] <- -99

# Replacing values in 'max_glu_serum' column
df$max_glu_serum[df$max_glu_serum == '>200'] <- 1
df$max_glu_serum[df$max_glu_serum == '>300'] <- 1
df$max_glu_serum[df$max_glu_serum == 'Norm'] <- 0
df$max_glu_serum[df$max_glu_serum == 'None'] <- -99
```

Handled age by converting the categorical age ranges into numeric values.

Since the dataset only provides age as 10-year categories, we approximated each patient's age by using the midpoint of their age category.

For example, for an age category of 20-30 years, we assumed the patient's age to be 25 years.

This transformation allows us to analyze the impact of age on readmission in a simplified, yet meaningful way.

```
# Loop to convert age intervals into numeric values (1-10)
for (i in 0:9) {
  # Create the age range in string format like '[0-10)', '[10-20)', etc.
  age_range <- paste0('[', 10 * i, '-', 10 * (i + 1), ')')

  # Replace the age range with numeric values (1 to 10)
  df$age[df$age == age_range] <- i + 1
}

# View the frequency of the updated 'age' column
table(df$age)
```

```
##
##     1    10     2     3     4     5     6     7     8     9
##    64  2594   466  1471  3538  9208 16546 21521 24815 16223
```

```
# Load necessary package
library(dplyr)
# Drop duplicates based on 'patient_nbr' and keep the first encounter
df2 <- df %>%
  distinct(patient_nbr, .keep_all = TRUE)

# Check the dimensions of the new dataset
dim(df2)  # Should be (70442, 55)
```

```
## [1] 67580    47
```

```
df %>% count(readmitted)
```

```
##   readmitted     n
## 1        <30 11066
## 2        >30 34649
## 3         NO 50731
```

Encoded the outcome variable to simplify the classification task into a binary problem.

The original dataset contains three categories for readmission: '< 30', '> 30', and 'No Readmission'.

To reduce this to a binary classification, we combined the '> 30' and 'No Readmission' categories into a single category.

This allows us to focus on predicting whether a patient is readmitted within 30 days or not.

```
df$readmitted[df$readmitted == ">30"] <- 0
df$readmitted[df$readmitted == "<30"] <- 1
df$readmitted[df$readmitted == "NO"]   <- 0
```

```
# Copying diagnosis columns
df$level1_diag1 <- df$diag_1
df$level2_diag1 <- df$diag_1
df$level1_diag2 <- df$diag_2
df$level2_diag2 <- df$diag_2
df$level1_diag3 <- df$diag_3
df$level2_diag3 <- df$diag_3

# Replacing 'V' or 'E' codes with 0
df$level1_diag1[grepl("V", df$diag_1) | grepl("E", df$diag_1)] <- 0
df$level2_diag1[grepl("V", df$diag_1) | grepl("E", df$diag_1)] <- 0

df$level1_diag2[grepl("V", df$diag_2) | grepl("E", df$diag_2)] <- 0
df$level2_diag2[grepl("V", df$diag_2) | grepl("E", df$diag_2)] <- 0

df$level1_diag3[grepl("V", df$diag_3) | grepl("E", df$diag_3)] <- 0
df$level2_diag3[grepl("V", df$diag_3) | grepl("E", df$diag_3)] <- 0

# Replacing '?' with -1
df$level1_diag1[df$level1_diag1 == "?"] <- -1
df$level2_diag1[df$level2_diag1 == "?"] <- -1
df$level1_diag2[df$level1_diag2 == "?"] <- -1
df$level2_diag2[df$level2_diag2 == "?"] <- -1
df$level1_diag3[df$level1_diag3 == "?"] <- -1
df$level2_diag3[df$level2_diag3 == "?"] <- -1

# Converting to numeric
df$level1_diag1 <- as.numeric(df$level1_diag1)
df$level2_diag1 <- as.numeric(df$level2_diag1)
df$level1_diag2 <- as.numeric(df$level1_diag2)
df$level2_diag2 <- as.numeric(df$level2_diag2)
```

```r
df$level1_diag3 <- as.numeric(df$level1_diag3)
df$level2_diag3 <- as.numeric(df$level2_diag3)
```

```r
# Ensure numeric type for diagnosis columns
df$level1_diag1 <- as.numeric(df$level1_diag1)
df$level1_diag2 <- as.numeric(df$level1_diag2)
df$level1_diag3 <- as.numeric(df$level1_diag3)

# Apply category mapping row-wise
for (i in 1:nrow(df)) {
  # Helper function for mapping
  map_diag <- function(x) {
    if ((x >= 390 & x < 460) | floor(x) == 785) {
      return(1)
    } else if ((x >= 460 & x < 520) | floor(x) == 786) {
      return(2)
    } else if ((x >= 520 & x < 580) | floor(x) == 787) {
      return(3)
    } else if (floor(x) == 250) {
      return(4)
    } else if (x >= 800 & x < 1000) {
      return(5)
    } else if (x >= 710 & x < 740) {
      return(6)
    } else if ((x >= 580 & x < 630) | floor(x) == 788) {
      return(7)
    } else if (x >= 140 & x < 240) {
      return(8)
    } else {
      return(0)
    }
  }
  df$level1_diag1[i] <- map_diag(df$level1_diag1[i])
  df$level1_diag2[i] <- map_diag(df$level1_diag2[i])
  df$level1_diag3[i] <- map_diag(df$level1_diag3[i])
}
```

```r
for (index in 1:nrow(df)) {
  # For level2_diag1
  if (df$level2_diag1[index] >= 390 & df$level2_diag1[index] < 399) {
    df$level2_diag1[index] <- 1
  } else if (df$level2_diag1[index] >= 401 & df$level2_diag1[index] < 415) {
    df$level2_diag1[index] <- 2
  } else if (df$level2_diag1[index] >= 415 & df$level2_diag1[index] < 460) {
    df$level2_diag1[index] <- 3
  } else if (floor(df$level2_diag1[index]) == 785) {
    df$level2_diag1[index] <- 4
  } else if (df$level2_diag1[index] >= 460 & df$level2_diag1[index] < 489) {
    df$level2_diag1[index] <- 5
  } else if (df$level2_diag1[index] >= 490 & df$level2_diag1[index] < 497) {
    df$level2_diag1[index] <- 6
  } else if (df$level2_diag1[index] >= 500 & df$level2_diag1[index] < 520) {
    df$level2_diag1[index] <- 7
```

```
} else if (floor(df$level2_diag1[index]) == 786) {
  df$level2_diag1[index] <- 8
} else if (df$level2_diag1[index] >= 520 & df$level2_diag1[index] < 530) {
  df$level2_diag1[index] <- 9
} else if (df$level2_diag1[index] >= 530 & df$level2_diag1[index] < 544) {
  df$level2_diag1[index] <- 10
} else if (df$level2_diag1[index] >= 550 & df$level2_diag1[index] < 554) {
  df$level2_diag1[index] <- 11
} else if (df$level2_diag1[index] >= 555 & df$level2_diag1[index] < 580) {
  df$level2_diag1[index] <- 12
} else if (floor(df$level2_diag1[index]) == 787) {
  df$level2_diag1[index] <- 13
} else if (floor(df$level2_diag1[index]) == 250) {
  df$level2_diag1[index] <- 14
} else if (df$level2_diag1[index] >= 800 & df$level2_diag1[index] < 1000) {
  df$level2_diag1[index] <- 15
} else if (df$level2_diag1[index] >= 710 & df$level2_diag1[index] < 740) {
  df$level2_diag1[index] <- 16
} else if (df$level2_diag1[index] >= 580 & df$level2_diag1[index] < 630) {
  df$level2_diag1[index] <- 17
} else if (floor(df$level2_diag1[index]) == 788) {
  df$level2_diag1[index] <- 18
} else if (df$level2_diag1[index] >= 140 & df$level2_diag1[index] < 240) {
  df$level2_diag1[index] <- 19
} else if (df$level2_diag1[index] >= 240 & df$level2_diag1[index] < 280 & floor(df$level2_diag1[index
  df$level2_diag1[index] <- 20
} else if (df$level2_diag1[index] >= 680 & df$level2_diag1[index] < 710 | floor(df$level2_diag1[index
  df$level2_diag1[index] <- 21
} else if (df$level2_diag1[index] >= 290 & df$level2_diag1[index] < 320) {
  df$level2_diag1[index] <- 22
} else {
  df$level2_diag1[index] <- 0
}

# For level2_diag2
if (df$level2_diag2[index] >= 390 & df$level2_diag2[index] < 399) {
  df$level2_diag2[index] <- 1
} else if (df$level2_diag2[index] >= 401 & df$level2_diag2[index] < 415) {
  df$level2_diag2[index] <- 2
} else if (df$level2_diag2[index] >= 415 & df$level2_diag2[index] < 460) {
  df$level2_diag2[index] <- 3
} else if (floor(df$level2_diag2[index]) == 785) {
  df$level2_diag2[index] <- 4
} else if (df$level2_diag2[index] >= 460 & df$level2_diag2[index] < 489) {
  df$level2_diag2[index] <- 5
} else if (df$level2_diag2[index] >= 490 & df$level2_diag2[index] < 497) {
  df$level2_diag2[index] <- 6
} else if (df$level2_diag2[index] >= 500 & df$level2_diag2[index] < 520) {
  df$level2_diag2[index] <- 7
} else if (floor(df$level2_diag2[index]) == 786) {
  df$level2_diag2[index] <- 8
} else if (df$level2_diag2[index] >= 520 & df$level2_diag2[index] < 530) {
  df$level2_diag2[index] <- 9
```

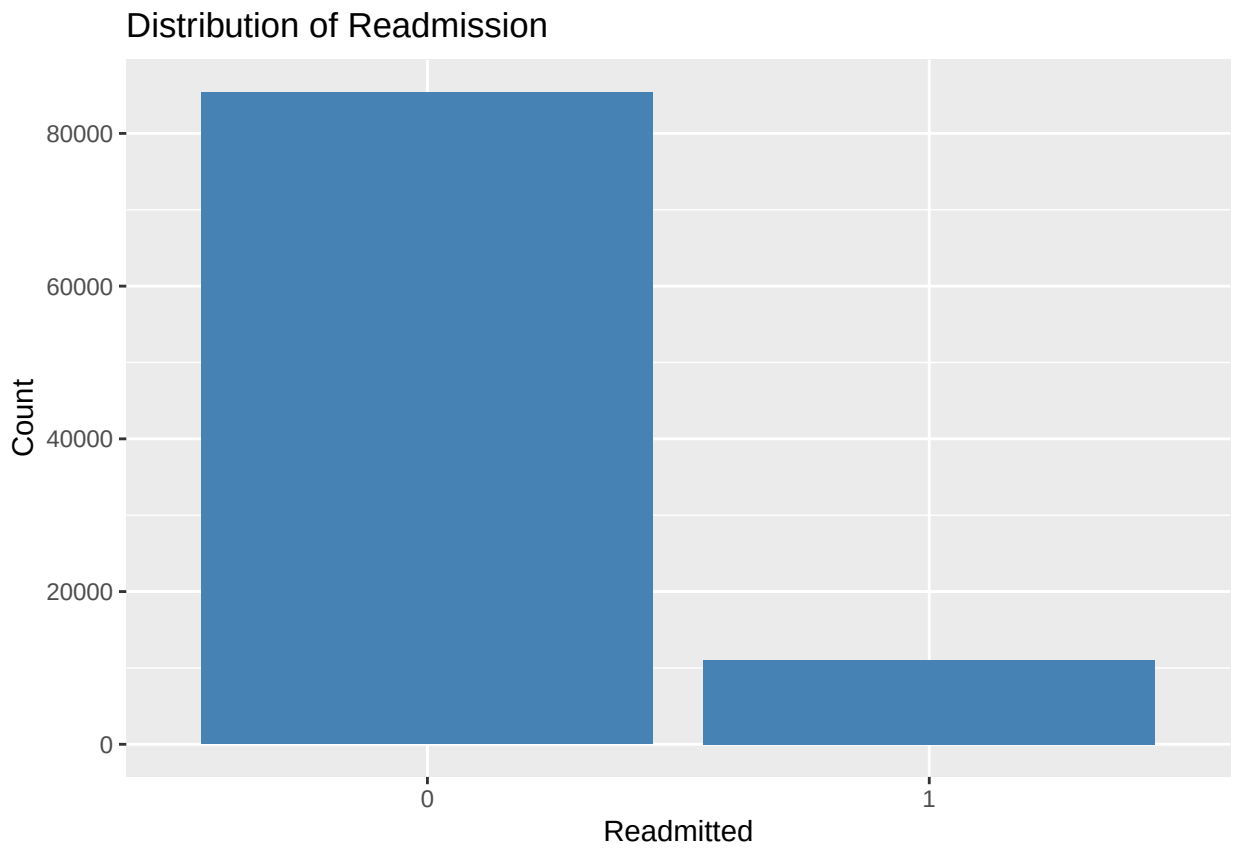
```
} else if (df$level2_diag2[index] >= 530 & df$level2_diag2[index] < 544) {
  df$level2_diag2[index] <- 10
} else if (df$level2_diag2[index] >= 550 & df$level2_diag2[index] < 554) {
  df$level2_diag2[index] <- 11
} else if (df$level2_diag2[index] >= 555 & df$level2_diag2[index] < 580) {
  df$level2_diag2[index] <- 12
} else if (floor(df$level2_diag2[index]) == 787) {
  df$level2_diag2[index] <- 13
} else if (floor(df$level2_diag2[index]) == 250) {
  df$level2_diag2[index] <- 14
} else if (df$level2_diag2[index] >= 800 & df$level2_diag2[index] < 1000) {
  df$level2_diag2[index] <- 15
} else if (df$level2_diag2[index] >= 710 & df$level2_diag2[index] < 740) {
  df$level2_diag2[index] <- 16
} else if (df$level2_diag2[index] >= 580 & df$level2_diag2[index] < 630) {
  df$level2_diag2[index] <- 17
} else if (floor(df$level2_diag2[index]) == 788) {
  df$level2_diag2[index] <- 18
} else if (df$level2_diag2[index] >= 140 & df$level2_diag2[index] < 240) {
  df$level2_diag2[index] <- 19
} else if (df$level2_diag2[index] >= 240 & df$level2_diag2[index] < 280 & floor(df$level2_diag2[index
  df$level2_diag2[index] <- 20
} else if (df$level2_diag2[index] >= 680 & df$level2_diag2[index] < 710 | floor(df$level2_diag2[index
  df$level2_diag2[index] <- 21
} else if (df$level2_diag2[index] >= 290 & df$level2_diag2[index] < 320) {
  df$level2_diag2[index] <- 22
} else {
  df$level2_diag2[index] <- 0
}

# For level2_diag3
if (df$level2_diag3[index] >= 390 & df$level2_diag3[index] < 399) {
  df$level2_diag3[index] <- 1
} else if (df$level2_diag3[index] >= 401 & df$level2_diag3[index] < 415) {
  df$level2_diag3[index] <- 2
} else if (df$level2_diag3[index] >= 415 & df$level2_diag3[index] < 460) {
  df$level2_diag3[index] <- 3
} else if (floor(df$level2_diag3[index]) == 785) {
  df$level2_diag3[index] <- 4
} else if (df$level2_diag3[index] >= 460 & df$level2_diag3[index] < 489) {
  df$level2_diag3[index] <- 5
} else if (df$level2_diag3[index] >= 490 & df$level2_diag3[index] < 497) {
  df$level2_diag3[index] <- 6
} else if (df$level2_diag3[index] >= 500 & df$level2_diag3[index] < 520) {
  df$level2_diag3[index] <- 7
} else if (floor(df$level2_diag3[index]) == 786) {
  df$level2_diag3[index] <- 8
} else if (df$level2_diag3[index] >= 520 & df$level2_diag3[index] < 530) {
  df$level2_diag3[index] <- 9
} else if (df$level2_diag3[index] >= 530 & df$level2_diag3[index] < 544) {
  df$level2_diag3[index] <- 10
} else if (df$level2_diag3[index] >= 550 & df$level2_diag3[index] < 554) {
  df$level2_diag3[index] <- 11
```

```
  } else if (df$level2_diag3[index] >= 555 & df$level2_diag3[index] < 580) {
    df$level2_diag3[index] <- 12
  } else if (floor(df$level2_diag3[index]) == 787) {
    df$level2_diag3[index] <- 13
  } else if (floor(df$level2_diag3[index]) == 250) {
    df$level2_diag3[index] <- 14
  } else if (df$level2_diag3[index] >= 800 & df$level2_diag3[index] < 1000) {
    df$level2_diag3[index] <- 15
  } else if (df$level2_diag3[index] >= 710 & df$level2_diag3[index] < 740) {
    df$level2_diag3[index] <- 16
  } else if (df$level2_diag3[index] >= 580 & df$level2_diag3[index] < 630) {
    df$level2_diag3[index] <- 17
  } else if (floor(df$level2_diag3[index]) == 788) {
    df$level2_diag3[index] <- 18
  } else if (df$level2_diag3[index] >= 140 & df$level2_diag3[index] < 240) {
    df$level2_diag3[index] <- 19
  } else if (df$level2_diag3[index] >= 240 & df$level2_diag3[index] < 280 & floor(df$level2_diag3[index
    df$level2_diag3[index] <- 20
  } else if (df$level2_diag3[index] >= 680 & df$level2_diag3[index] < 710 | floor(df$level2_diag3[index
    df$level2_diag3[index] <- 21
  } else if (df$level2_diag3[index] >= 290 & df$level2_diag3[index] < 320) {
    df$level2_diag3[index] <- 22
  } else {
    df$level2_diag3[index] <- 0
  }
}
```

##data visualization

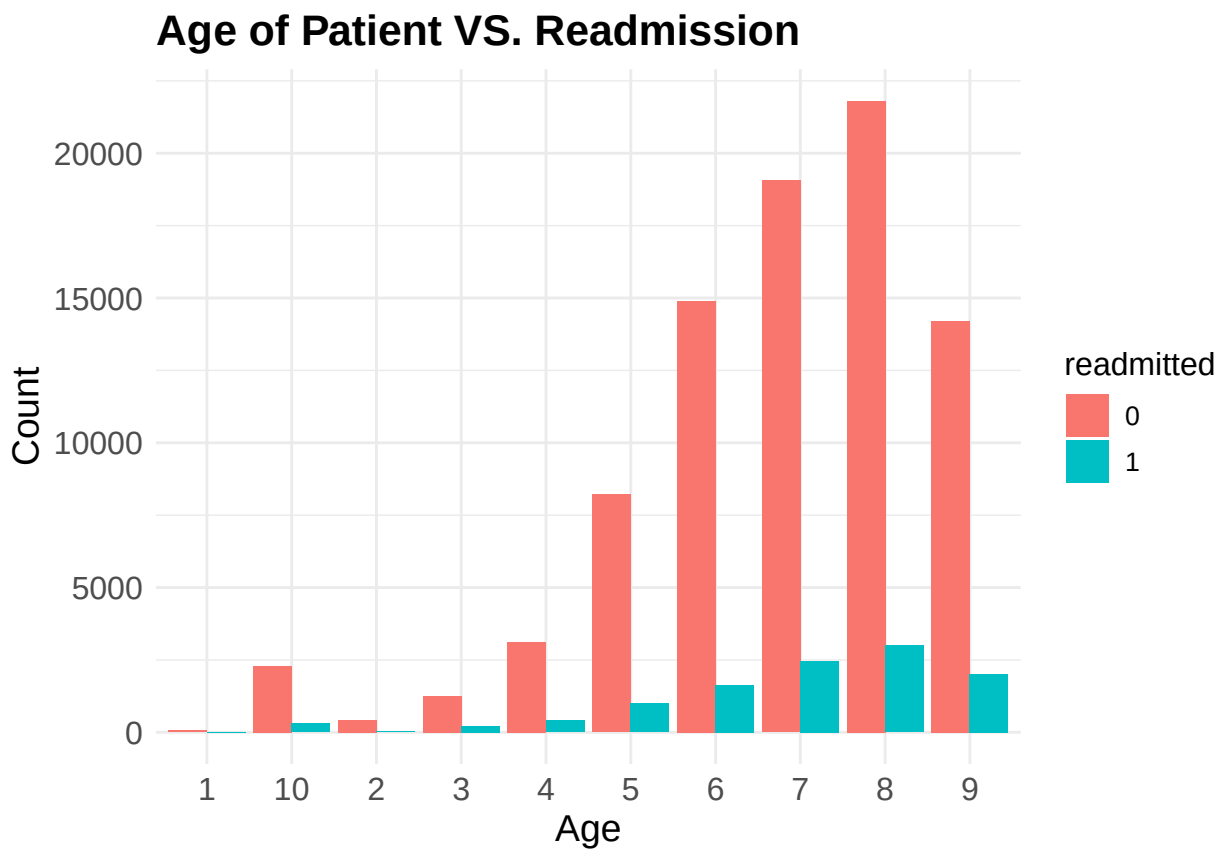
#Distribution of Readmission #The target variable is imbalance. Number of readmitted patient are quite less as compared to Not readmitted

```
library(ggplot2)
ggplot(df, aes(x = readmitted)) +
  geom_bar(fill = "steelblue") +
  ggtitle("Distribution of Readmission") +
  xlab("Readmitted") +
  ylab("Count")
```



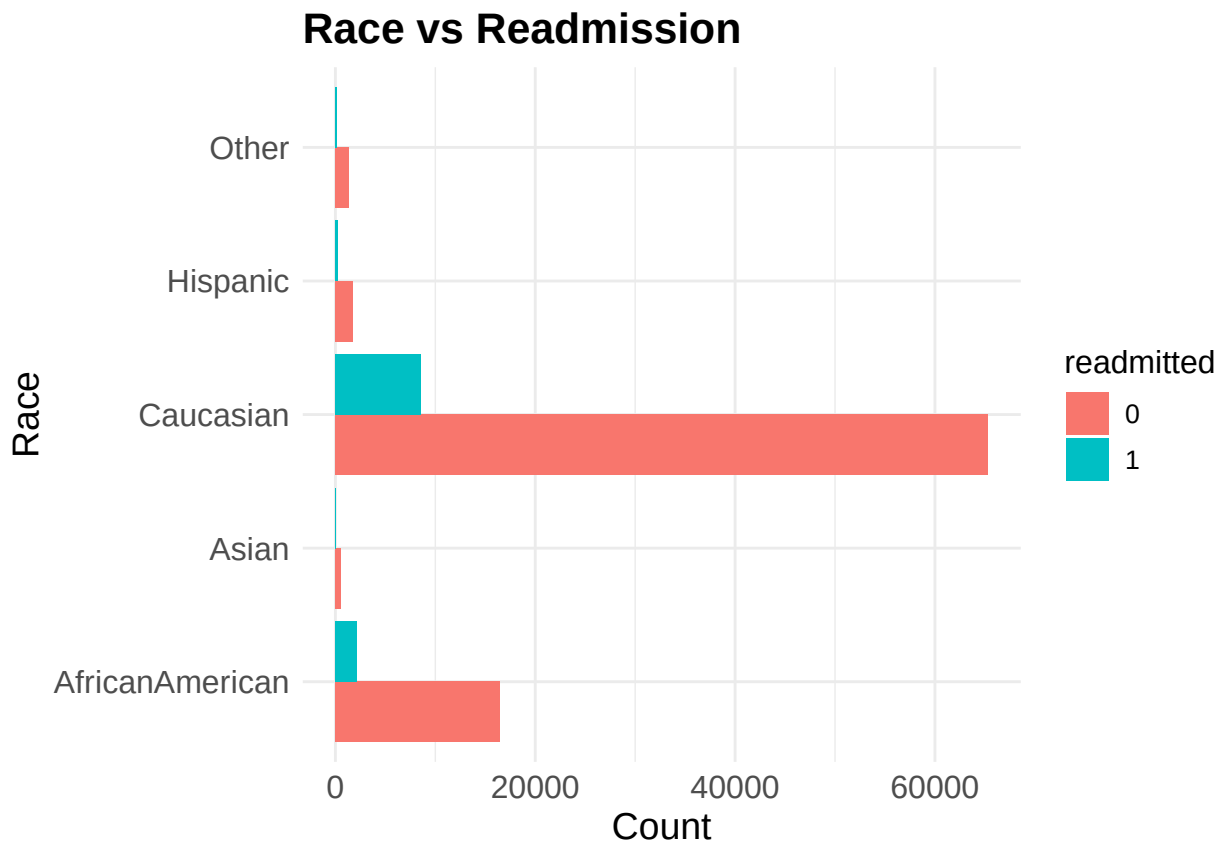
#Age and Readmission

```
library(ggplot2)

# Plot with age on y-axis and readmission as fill (equivalent to hue in seaborn)
ggplot(df, aes(y = age, fill = readmitted)) +
  geom_bar(position = "dodge") +
  ggtitle("Age of Patient VS. Readmission") +
  xlab("Count") +
  ylab("Age") +
  theme_minimal() +
  theme(plot.title = element_text(size = 16, face = "bold"),
        axis.text = element_text(size = 12),
        axis.title = element_text(size = 14)) +
  theme(legend.title = element_text(size = 12),
        legend.text = element_text(size = 10)) +
  coord_flip()  # Flip axes for horizontal bars (like y=... in seaborn)
```



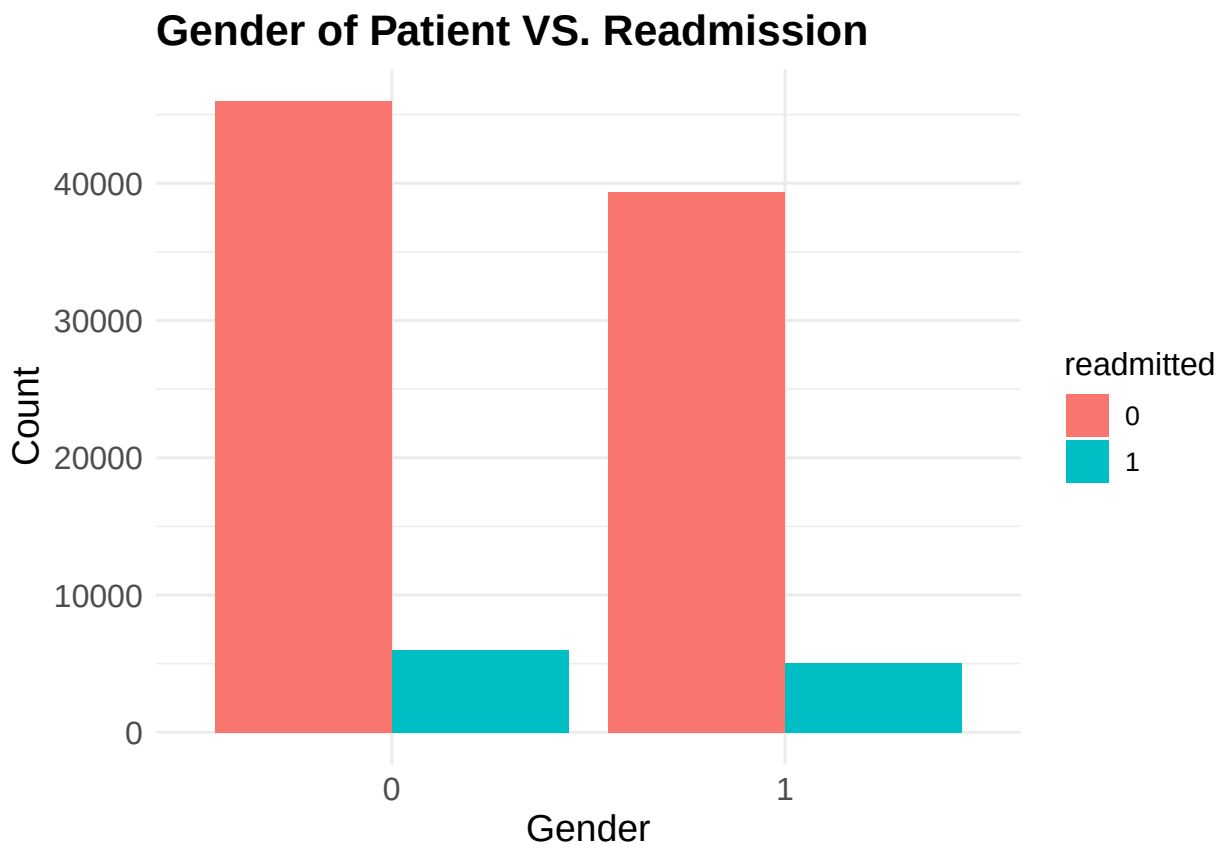
#Ethnicity of patient and Readmission

```
ggplot(df, aes(y = race, fill = readmitted)) +
  geom_bar(position = "dodge") +
  ggtitle("Race vs Readmission") +
  xlab("Count") +
  ylab("Race") +
  theme_minimal() +
  theme(plot.title = element_text(size = 16, face = "bold"),
        axis.text = element_text(size = 12),
        axis.title = element_text(size = 14),
        legend.title = element_text(size = 12),
        legend.text = element_text(size = 10))
```



##Gender and Readmission ##Male = 1 ##Female = 0

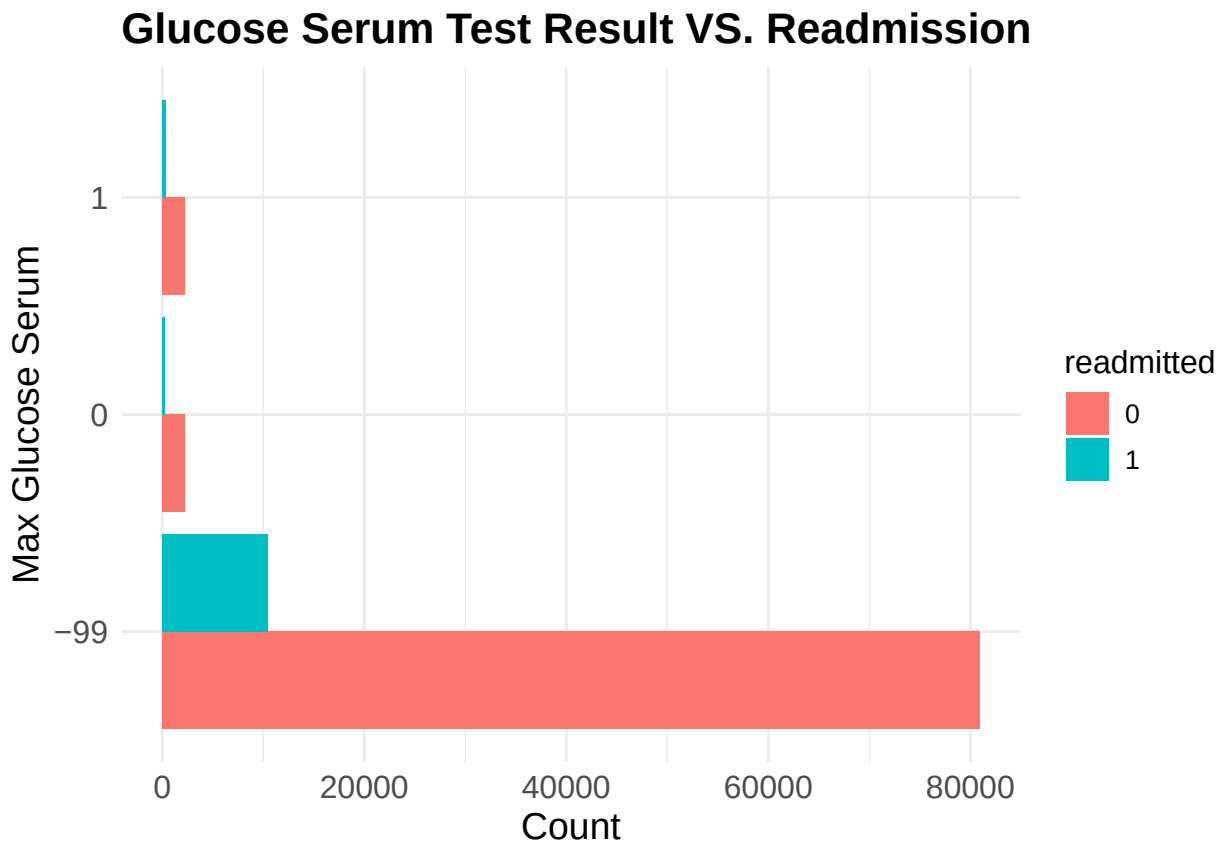
```
ggplot(df, aes(x = gender, fill = readmitted)) +
  geom_bar(position = "dodge") +
  ggtitle("Gender of Patient VS. Readmission") +
  xlab("Gender") +
  ylab("Count") +
  theme_minimal() +
  theme(plot.title = element_text(size = 16, face = "bold"),
        axis.text = element_text(size = 12),
        axis.title = element_text(size = 14),
        legend.title = element_text(size = 12),
        legend.text = element_text(size = 10))
```



##Glucose serum test result and Readmission ##Glucose Serum test - A blood glucose test is used to find out if your blood sugar levels are in the healthy range. It is often used to help diagnose and monitor diabetes.

##'>200' : 1 = indicates diabetes ##'>300' : 1 = Indicates diabetes ##'Norm' : 0 = Normal ##'None' : -99 = test was not taken

```
ggplot(df, aes(y = max_glu_serum, fill = readmitted)) +
  geom_bar(position = "dodge") +
  ggtitle("Glucose Serum Test Result VS. Readmission") +
  xlab("Count") +
  ylab("Max Glucose Serum") +
  theme_minimal() +
  theme(plot.title = element_text(size = 16, face = "bold"),
        axis.text = element_text(size = 12),
        axis.title = element_text(size = 14),
        legend.title = element_text(size = 12),
        legend.text = element_text(size = 10))
```



##data preprocessing for modelling

```
# Convert age column to integer
df$age <- as.integer(df$age)
print(table(df$age))
```

```
##
##     1     2     3     4     5     6     7     8     9    10
##    64   466  1471  3538  9208 16546 21521 24815 16223  2594
```

```
# Convert age categories to mid-point values
age_dict <- setNames(c(5, 15, 25, 35, 45, 55, 65, 75, 85, 95),
                     c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10))
df$age <- age_dict[df$age]
print(table(df$age))
```

```
##
##     5    15    25    35    45    55    65    75    85    95
##    64   466  1471  3538  9208 16546 21521 24815 16223  2594
```

```
# Convert nominal features to factor type (equivalent to 'object' type in pandas)
nominal_cols <- c('encounter_id', 'patient_nbr', 'gender', 'admission_type_id', 'discharge_disposition_
                  'admission_source_id', 'A1Cresult', 'metformin', 'repaglinide', 'nateglinide',
                  'chlorpropamide', 'glimepiride', 'acetohexamide', 'glipizide', 'glyburide',
                  'tolbutamide', 'pioglitazone', 'rosiglitazone', 'acarbose', 'miglitol', 'troglitazone',
```

```
                'tolazamide', 'insulin', 'glyburide.metformin', 'glipizide.metformin',
                'glimepiride.pioglitazone', 'metformin.rosiglitazone', 'metformin.pioglitazone',
                'change', 'diabetesMed', 'age', 'max_glu_serum', 'level1_diag1',
                'level1_diag2', 'level1_diag3', 'level2_diag1', 'level2_diag2', 'level2_diag3')

df[nominal_cols] <- lapply(df[nominal_cols], as.factor)
str(df)
```

```
## 'data.frame':    96446 obs. of  53 variables:
##  $ encounter_id            : Factor w/ 96446 levels "12522","15738",..: 17 12 42 3 5 9 11 1 2 4 ...
##  $ patient_nbr             : Factor w/ 67580 levels "135","378","729",..: 35703 49317 46679 30009 46
##  $ race                    : chr  "Caucasian" "AfricanAmerican" "Caucasian" "Caucasian" ...
##  $ gender                  : Factor w/ 2 levels "0","1": 1 1 2 2 2 2 2 1 1 1 ...
##  $ age                     : Factor w/ 10 levels "5","15","25",..: 2 3 4 5 6 7 8 9 10 5 ...
##  $ admission_type_id       : Factor w/ 4 levels "1","3","4","5": 1 1 1 1 1 2 1 1 2 1 ...
##  $ discharge_disposition_id: Factor w/ 9 levels "1","2","7","10",..: 1 1 1 1 1 1 1 1 2 1 ...
##  $ admission_source_id     : Factor w/ 6 levels "1","4","7","8",..: 3 3 3 3 1 1 3 2 2 3 ...
##  $ time_in_hospital        : int  3 2 2 1 3 4 5 13 12 9 ...
##  $ num_lab_procedures      : int  59 11 44 51 31 70 73 68 33 47 ...
##  $ num_procedures          : int  0 5 1 0 6 1 0 2 3 2 ...
##  $ num_medications         : int  18 13 16 8 16 21 12 28 18 17 ...
##  $ number_outpatient       : int  0 2 0 0 0 0 0 0 0 0 ...
##  $ number_emergency        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ number_inpatient        : int  0 1 0 0 0 0 0 0 0 0 ...
##  $ diag_1                  : chr  "276" "648" "8" "197" ...
##  $ diag_2                  : chr  "250.01" "250" "250.43" "157" ...
##  $ diag_3                  : chr  "255" "V27" "403" "250" ...
##  $ number_diagnoses        : int  9 6 7 5 9 7 8 8 8 9 ...
##  $ max_glu_serum           : Factor w/ 3 levels "-99","0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ A1Cresult               : Factor w/ 3 levels "-99","0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ metformin               : Factor w/ 2 levels "0","1": 1 1 1 1 1 2 1 1 1 1 ...
##  $ repaglinide             : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ nateglinide             : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ chlorpropamide          : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ glimepiride             : Factor w/ 2 levels "0","1": 1 1 1 1 1 2 1 1 1 1 ...
##  $ acetohexamide           : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ glipizide               : Factor w/ 2 levels "0","1": 1 2 1 2 1 1 1 2 1 1 ...
##  $ glyburide               : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 2 1 1 1 ...
##  $ tolbutamide             : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ pioglitazone            : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ rosiglitazone           : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 2 1 ...
##  $ acarbose                : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ miglitol                : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ troglitazone            : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ tolazamide              : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ insulin                 : Factor w/ 2 levels "0","1": 2 1 2 2 2 2 1 2 2 2 ...
##  $ glyburide.metformin     : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ glipizide.metformin     : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ glimepiride.pioglitazone: Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ metformin.rosiglitazone : Factor w/ 1 level "0": 1 1 1 1 1 1 1 1 1 1 ...
##  $ metformin.pioglitazone  : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ change                  : Factor w/ 2 levels "0","1": 2 1 2 2 1 2 1 2 2 1 ...
##  $ diabetesMed             : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
```

```
##  $ readmitted              : chr  "0" "0" "0" "0" ...
##  $ service_utilization     : int  0 3 0 0 0 0 0 0 0 0 ...
##  $ numchange               : num  1 0 1 0 0 0 0 0 0 0 ...
##  $ level1_diag1            : Factor w/ 9 levels "0","1","2","3",..: 1 1 1 9 2 2 2 2 2 5 ...
##  $ level2_diag1            : Factor w/ 23 levels "0","1","2","3",..: 21 1 1 20 3 3 4 2 4 15 ...
##  $ level1_diag2            : Factor w/ 9 levels "0","1","2","3",..: 5 5 5 9 2 2 3 2 9 2 ...
##  $ level2_diag2            : Factor w/ 23 levels "0","1","2","3",..: 15 15 15 20 3 3 7 4 20 3 ...
##  $ level1_diag3            : Factor w/ 9 levels "0","1","2","3",..: 1 1 2 5 5 1 5 1 3 6 ...
##  $ level2_diag3            : Factor w/ 23 levels "0","1","2","3",..: 21 1 3 15 15 1 15 1 6 16 ...
```

```r
# Initialize nummed column with zeros
df$nummed <- 0

# Sum medication columns
for (col in keys) {
  df$nummed <- df$nummed + as.numeric(as.character(df[[col]]))
}
table(df$nummed)
```

```
##
##     0     1     2     3     4     5     6
## 22156 44589 20901  7448  1290    57     5
```

```r
# Get list of only numeric features (excluding 'readmitted')
num_cols <- setdiff(names(df)[sapply(df, is.numeric)], 'readmitted')
num_cols
```

```
##  [1] "time_in_hospital"    "num_lab_procedures"  "num_procedures"
##  [4] "num_medications"     "number_outpatient"   "number_emergency"
##  [7] "number_inpatient"    "number_diagnoses"    "service_utilization"
## [10] "numchange"           "nummed"
```

```r
# Create dataframe for statistics and log transformations
statdataframe <- data.frame(numeric_column = num_cols)
skew_before <- numeric(length(num_cols))
skew_after <- numeric(length(num_cols))
kurt_before <- numeric(length(num_cols))
kurt_after <- numeric(length(num_cols))
standard_deviation_before <- numeric(length(num_cols))
standard_deviation_after <- numeric(length(num_cols))
log_transform_needed <- character(length(num_cols))
log_type <- character(length(num_cols))

# Calculate statistics and determine transformation needs
for (i in 1:length(num_cols)) {
  col_name <- num_cols[i]

  # Calculate skewness
  skewval <- moments::skewness(df[[col_name]], na.rm = TRUE)
  skew_before[i] <- skewval

  # Calculate kurtosis
```

```r
  kurtval <- moments::kurtosis(df[[col_name]], na.rm = TRUE) - 3  # R kurtosis is different from Python
  kurt_before[i] <- kurtval

  # Calculate standard deviation
  sdval <- sd(df[[col_name]], na.rm = TRUE)
  standard_deviation_before[i] <- sdval

  if ((abs(skewval) > 2) & (abs(kurtval) > 2)) {
    log_transform_needed[i] <- "Yes"

    # Determine log type based on zero percentage
    if (sum(df[[col_name]] == 0, na.rm = TRUE) / nrow(df) <= 0.02) {
      log_type[i] <- "log"

      # Calculate transformed statistics
      temp_data <- df[train_data[[col_name]] > 0, col_name]
      skewvalnew <- moments::skewness(log(temp_data), na.rm = TRUE)
      skew_after[i] <- skewvalnew

      kurtvalnew <- moments::kurtosis(log(temp_data), na.rm = TRUE) - 3  # Adjusting for R kurtosis
      kurt_after[i] <- kurtvalnew

      sdvalnew <- sd(log(temp_data), na.rm = TRUE)
      standard_deviation_after[i] <- sdvalnew

    } else {
      log_type[i] <- "log1p"

      # Calculate transformed statistics
      temp_data <- df[df[[col_name]] >= 0, col_name]
      skewvalnew <- moments::skewness(log1p(temp_data), na.rm = TRUE)
      skew_after[i] <- skewvalnew

      kurtvalnew <- moments::kurtosis(log1p(temp_data), na.rm = TRUE) - 3
      kurt_after[i] <- kurtvalnew

      sdvalnew <- sd(log1p(temp_data), na.rm = TRUE)
      standard_deviation_after[i] <- sdvalnew
    }
  } else {
    log_type[i] <- "NA"
    log_transform_needed[i] <- "No"

    skew_after[i] <- skewval
    kurt_after[i] <- kurtval
    standard_deviation_after[i] <- sdval
  }
}

# Assemble statistics dataframe
statdataframe$skew_before <- skew_before
statdataframe$kurtosis_before <- kurt_before
statdataframe$standard_deviation_before <- standard_deviation_before
```

```r
statdataframe$log_transform_needed <- log_transform_needed
statdataframe$log_type <- log_type
statdataframe$skew_after <- skew_after
statdataframe$kurtosis_after <- kurt_after
statdataframe$standard_deviation_after <- standard_deviation_after
statdataframe
```

```
##         numeric_column skew_before kurtosis_before standard_deviation_before
## 1     time_in_hospital   1.1274929       0.8389447                 2.9823302
## 2   num_lab_procedures  -0.2406224      -0.2533244                19.6567817
## 3       num_procedures   1.3132158       0.8559933                 1.7031834
## 4      num_medications   1.3391662       3.5490792                 8.0725162
## 5    number_outpatient   8.7673530     146.2373172                 1.2800608
## 6     number_emergency  22.6955676    1165.0799352                 0.9480890
## 7     number_inpatient   3.5662140      20.0437113                 1.2699746
## 8     number_diagnoses  -0.8077283      -0.3726008                 1.8366590
## 9  service_utilization   5.3122916      67.1904723                 2.3157889
## 10           numchange   1.4265253       1.4517608                 0.4886143
## 11              nummed   0.6769720       0.2770472                 0.9233360
##    log_transform_needed log_type skew_after kurtosis_after
## 1                    No       NA  1.1274929      0.8389447
## 2                    No       NA -0.2406224     -0.2533244
## 3                    No       NA  1.3132158      0.8559933
## 4                    No       NA  1.3391662      3.5490792
## 5                   Yes    log1p  2.7085848      7.6480759
## 6                   Yes    log1p  3.6144147     15.8532211
## 7                   Yes    log1p  1.4251046      1.3190553
## 8                    No       NA -0.8077283     -0.3726008
## 9                   Yes    log1p  1.0972272      0.4971530
## 10                   No       NA  1.4265253      1.4517608
## 11                   No       NA  0.6769720      0.2770472
##    standard_deviation_after
## 1                 2.9823302
## 2                19.6567817
## 3                 1.7031834
## 4                 8.0725162
## 5                 0.4329489
## 6                 0.3187416
## 7                 0.5133891
## 8                 1.8366590
## 9                 0.6656560
## 10                0.4886143
## 11                0.9233360
```

```r
# Perform log transformations based on previous analysis
for (i in 1:nrow(statdataframe)) {
  if (statdataframe$log_transform_needed[i] == "Yes") {
    colname <- as.character(statdataframe$numeric_column[i])

    if (statdataframe$log_type[i] == "log") {
      df <- df[df[[colname]] > 0, ]
      df[[paste0(colname, "_log")]] <- log(df[[colname]])
```

```
    } else if (statdataframe$log_type[i] == "log1p") {
      df <- df[df[[colname]] >= 0, ]
      df[[paste0(colname, "_log1p")]] <- log1p(df[[colname]])
    }
  }
}

# Drop specific columns
df <- df[, !names(df) %in% c('number_outpatient', 'number_inpatient', 'number_emergency', 'service_utili
dim(df)
```

```
## [1] 96446    54
```

```
# Get numeric features excluding 'readmitted'
numerics <- setdiff(names(df)[sapply(df, is.numeric)], 'readmitted')
numerics
```

```
##  [1] "time_in_hospital"          "num_lab_procedures"
##  [3] "num_procedures"            "num_medications"
##  [5] "number_diagnoses"          "numchange"
##  [7] "nummed"                    "number_outpatient_log1p"
##  [9] "number_emergency_log1p"    "number_inpatient_log1p"
## [11] "service_utilization_log1p"
```

```
# Convert specific columns to integer
df$encounter_id <- as.integer(as.character(df$encounter_id))
df$patient_nbr <- as.integer(as.character(df$patient_nbr))
df$diabetesMed <- as.integer(as.character(df$diabetesMed))
df$change <- as.integer(as.character(df$change))

# Convert medication columns to integer
med_cols <- c('metformin', 'repaglinide', 'nateglinide', 'chlorpropamide', 'glimepiride', 'acetohexamid
              'glipizide', 'glyburide', 'tolbutamide', 'pioglitazone', 'rosiglitazone', 'acarbose', 'mig
              'troglitazone', 'tolazamide', 'insulin', 'glyburide.metformin', 'glipizide.metformin',
              'glimepiride.pioglitazone', 'metformin.rosiglitazone', 'metformin.pioglitazone', 'A1Cresul

df[med_cols] <- lapply(df[med_cols], function(x) as.integer(as.character(x)))
str(df)
```

```
## 'data.frame':    96446 obs. of  54 variables:
##  $ encounter_id             : int  149190 64410 500364 16680 35754 55842 63768 12522 15738 28236 ...
##  $ patient_nbr              : int  55629189 86047875 82442376 42519267 82637451 84259809 114882984 48
##  $ race                     : chr  "Caucasian" "AfricanAmerican" "Caucasian" "Caucasian" ...
##  $ gender                   : Factor w/ 2 levels "0","1": 1 1 2 2 2 2 2 1 1 1 ...
##  $ age                      : Factor w/ 10 levels "5","15","25",..: 2 3 4 5 6 7 8 9 10 5 ...
##  $ admission_type_id        : Factor w/ 4 levels "1","3","4","5": 1 1 1 1 1 2 1 1 2 1 ...
##  $ discharge_disposition_id : Factor w/ 9 levels "1","2","7","10",..: 1 1 1 1 1 1 1 1 2 1 ...
##  $ admission_source_id      : Factor w/ 6 levels "1","4","7","8",..: 3 3 3 3 1 1 3 2 2 3 ...
##  $ time_in_hospital         : int  3 2 2 1 3 4 5 13 12 9 ...
##  $ num_lab_procedures       : int  59 11 44 51 31 70 73 68 33 47 ...
##  $ num_procedures           : int  0 5 1 0 6 1 0 2 3 2 ...
##  $ num_medications          : int  18 13 16 8 16 21 12 28 18 17 ...
```

```
##  $ diag_1                   : chr  "276" "648" "8" "197" ...
##  $ diag_2                   : chr  "250.01" "250" "250.43" "157" ...
##  $ diag_3                   : chr  "255" "V27" "403" "250" ...
##  $ number_diagnoses         : int  9 6 7 5 9 7 8 8 8 9 ...
##  $ max_glu_serum            : Factor w/ 3 levels "-99","0","1": 1 1 1 1 1 1 1 1 1 ...
##  $ A1Cresult                : int  -99 -99 -99 -99 -99 -99 -99 -99 -99 -99 ...
##  $ metformin                : int  0 0 0 0 0 1 0 0 0 0 ...
##  $ repaglinide              : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nateglinide              : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ chlorpropamide           : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ glimepiride              : int  0 0 0 0 0 1 0 0 0 0 ...
##  $ acetohexamide            : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ glipizide                : int  0 1 0 1 0 0 0 1 0 0 ...
##  $ glyburide                : int  0 0 0 0 0 0 1 0 0 0 ...
##  $ tolbutamide              : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ pioglitazone             : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ rosiglitazone            : int  0 0 0 0 0 0 0 0 1 0 ...
##  $ acarbose                 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ miglitol                 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ troglitazone             : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ tolazamide               : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ insulin                  : int  1 0 1 1 1 1 0 1 1 1 ...
##  $ glyburide.metformin      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ glipizide.metformin      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ glimepiride.pioglitazone : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ metformin.rosiglitazone  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ metformin.pioglitazone   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ change                   : int  1 0 1 1 0 1 0 1 1 0 ...
##  $ diabetesMed              : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ readmitted               : chr  "0" "0" "0" "0" ...
##  $ numchange                : num  1 0 1 0 0 0 0 0 0 0 ...
##  $ level1_diag1             : Factor w/ 9 levels "0","1","2","3",..: 1 1 1 9 2 2 2 2 2 5 ...
##  $ level2_diag1             : Factor w/ 23 levels "0","1","2","3",..: 21 1 1 20 3 3 4 2 4 15 ...
##  $ level1_diag2             : Factor w/ 9 levels "0","1","2","3",..: 5 5 5 9 2 2 3 2 9 2 ...
##  $ level2_diag2             : Factor w/ 23 levels "0","1","2","3",..: 15 15 15 20 3 3 7 4 20 3 ...
##  $ level1_diag3             : Factor w/ 9 levels "0","1","2","3",..: 1 1 2 5 5 1 5 1 3 6 ...
##  $ level2_diag3             : Factor w/ 23 levels "0","1","2","3",..: 21 1 3 15 15 1 15 1 6 16 ...
##  $ nummed                   : num  1 1 1 2 1 3 1 2 2 1 ...
##  $ number_outpatient_log1p  : num  0 1.1 0 0 0 ...
##  $ number_emergency_log1p   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ number_inpatient_log1p   : num  0 0.693 0 0 0 ...
##  $ service_utilization_log1p: num  0 1.39 0 0 0 ...
```

```r
# Create a deep copy
dfcopy <- df

# Transform readmitted variable
df$readmitted <- ifelse(df$readmitted == 2, 0, df$readmitted)

# Drop columns
df <- df[, !names(df) %in% c('diag_1', 'diag_2', 'diag_3', 'level2_diag1', 'level1_diag2',
                             'level2_diag2', 'level1_diag3', 'level2_diag3')]

# Create interaction terms
```

```r
interactionterms <- list(
  c('num_medications', 'time_in_hospital'),
  c('num_medications', 'num_procedures'),
  c('time_in_hospital', 'num_lab_procedures'),
  c('num_medications', 'num_lab_procedures'),
  c('num_medications', 'number_diagnoses'),
  c('change', 'num_medications'),
  c('number_diagnoses', 'time_in_hospital'),
  c('num_medications', 'numchange')
)

for (inter in interactionterms) {
  name <- paste(inter[1], inter[2], sep = "|")
  df[[name]] <- df[[inter[1]]] * df[[inter[2]]]
}

df$age <- as.numeric(as.character(df$age))
df[["age|number_diagnoses"]] <- df$age * df$number_diagnoses


head(df[c('num_medications', 'time_in_hospital', 'num_medications|time_in_hospital')])
```

```
##   num_medications time_in_hospital num_medications|time_in_hospital
## 2              18                3                                54
## 3              13                2                                26
## 4              16                2                                32
## 5               8                1                                 8
## 6              16                3                                48
## 7              21                4                                84
```

```r
# Feature scaling statistics
datf <- data.frame(features = numerics)
datf$std_dev <- sapply(datf$features, function(x) sd(df[[x]], na.rm = TRUE))
datf$mean <- sapply(datf$features, function(x) mean(df[[x]], na.rm = TRUE))

# Keep first encounter per patient
df2 <- df[!duplicated(df$patient_nbr), ]
dim(df2)
```

```
## [1] 67580    55
```

```r
# Standardize function
standardize <- function(raw_data) {
  return((raw_data - colMeans(raw_data, na.rm = TRUE)) / apply(raw_data, 2, sd, na.rm = TRUE))
}

# Apply standardization to numeric columns
df2[numerics] <- standardize(df2[numerics])

# Remove outliers with z-score > 3
z_scores <- apply(df2[numerics], 2, scale)
df2 <- df2[apply(abs(z_scores) < 3, 1, all), ]
```

```r
# Convert level1_diag1 to factor
df2$level1_diag1 <- as.factor(df2$level1_diag1)

# Create dummy variables
library(fastDummies)
df_pd <- dummy_cols(df2,
                    select_columns = c('gender', 'admission_type_id', 'discharge_disposition_id',
                                       'admission_source_id', 'max_glu_serum', 'A1Cresult', 'level1_diag1
                    remove_first_dummy = TRUE)

# Create race dummies separately and join
race_dummies <- dummy_cols(df_pd['race'], remove_selected_columns = TRUE)
df_pd <- cbind(df_pd[, !names(df_pd) %in% c('race')], race_dummies)

# Define column groups
non_num_cols <- c('race', 'gender', 'admission_type_id', 'discharge_disposition_id', 'admission_source_
                  'max_glu_serum', 'A1Cresult', 'level1_diag1')
num_cols <- setdiff(names(df)[sapply(df, is.numeric)], c('readmitted', 'change'))
num_cols
```

```
##  [1] "encounter_id"                      "patient_nbr"
##  [3] "age"                               "time_in_hospital"
##  [5] "num_lab_procedures"                "num_procedures"
##  [7] "num_medications"                   "number_diagnoses"
##  [9] "A1Cresult"                         "metformin"
## [11] "repaglinide"                       "nateglinide"
## [13] "chlorpropamide"                    "glimepiride"
## [15] "acetohexamide"                     "glipizide"
## [17] "glyburide"                         "tolbutamide"
## [19] "pioglitazone"                      "rosiglitazone"
## [21] "acarbose"                          "miglitol"
## [23] "troglitazone"                      "tolazamide"
## [25] "insulin"                           "glyburide.metformin"
## [27] "glipizide.metformin"               "glimepiride.pioglitazone"
## [29] "metformin.rosiglitazone"           "metformin.pioglitazone"
## [31] "diabetesMed"                       "numchange"
## [33] "nummed"                            "number_outpatient_log1p"
## [35] "number_emergency_log1p"            "number_inpatient_log1p"
## [37] "service_utilization_log1p"         "num_medications|time_in_hospital"
## [39] "num_medications|num_procedures"    "time_in_hospital|num_lab_procedures"
## [41] "num_medications|num_lab_procedures" "num_medications|number_diagnoses"
## [43] "change|num_medications"            "number_diagnoses|time_in_hospital"
## [45] "num_medications|numchange"         "age|number_diagnoses"
```

```r
# Find new dummy column names
new_non_num_cols <- character(0)
for (i in non_num_cols) {
  for (j in names(df_pd)) {
    if (grepl(i, j)) {
      new_non_num_cols <- c(new_non_num_cols, j)
    }
  }
```

```
}
new_non_num_cols
```

```
##  [1] "race_AfricanAmerican"        "race_Asian"
##  [3] "race_Caucasian"              "race_Hispanic"
##  [5] "race_Other"                  "gender"
##  [7] "gender_1"                    "admission_type_id"
##  [9] "admission_type_id_3"         "admission_type_id_4"
## [11] "admission_type_id_5"         "discharge_disposition_id"
## [13] "discharge_disposition_id_2"  "discharge_disposition_id_7"
## [15] "discharge_disposition_id_10" "discharge_disposition_id_18"
## [17] "discharge_disposition_id_19" "discharge_disposition_id_20"
## [19] "discharge_disposition_id_27" "discharge_disposition_id_28"
## [21] "admission_source_id"         "admission_source_id_4"
## [23] "admission_source_id_7"       "admission_source_id_8"
## [25] "admission_source_id_9"       "admission_source_id_11"
## [27] "max_glu_serum"               "max_glu_serum_0"
## [29] "max_glu_serum_1"             "A1Cresult"
## [31] "A1Cresult_0"                 "A1Cresult_1"
## [33] "level1_diag1"                "level1_diag1_1"
## [35] "level1_diag1_2"              "level1_diag1_3"
## [37] "level1_diag1_4"              "level1_diag1_5"
## [39] "level1_diag1_6"              "level1_diag1_7"
## [41] "level1_diag1_8"
```

```
# Find interaction terms
l <- character(0)
for (feature in names(df_pd)) {
  if (grepl("\\|", feature)) {
    l <- c(l, feature)
  }
}
l
```

```
## [1] "num_medications|time_in_hospital"    "num_medications|num_procedures"
## [3] "time_in_hospital|num_lab_procedures" "num_medications|num_lab_procedures"
## [5] "num_medications|number_diagnoses"    "change|num_medications"
## [7] "number_diagnoses|time_in_hospital"   "num_medications|numchange"
## [9] "age|number_diagnoses"
```

##modelling
##2) Can logistic regression, a generalized linear model (GLM), provide reliable predictions for hospital readmission outcomes? ##logistic regression

```
feature_set <- c('age', 'time_in_hospital', 'num_procedures', 'num_medications', 'number_outpatient_log
                 'number_emergency_log1p', 'number_inpatient_log1p', 'number_diagnoses', 'metformin',
                 'repaglinide', 'nateglinide', 'chlorpropamide', 'glimepiride', 'glipizide', 'glyburide',
                 'pioglitazone', 'rosiglitazone', 'acarbose', 'tolazamide', 'insulin', 'glyburide.metformi
                 'race_AfricanAmerican', 'race_Asian', 'race_Caucasian', 'race_Hispanic', 'race_Other', 'g
                 'admission_type_id_3', 'admission_type_id_5', 'discharge_disposition_id_2', 'discharge_di
                 'discharge_disposition_id_10', 'discharge_disposition_id_18', 'admission_source_id_4',
                 'admission_source_id_7', 'admission_source_id_9', 'max_glu_serum_0', 'max_glu_serum_1',
```

```
               'A1Cresult_1', 'num_medications|time_in_hospital', 'num_medications|num_procedures',
               'time_in_hospital|num_lab_procedures', 'num_medications|num_lab_procedures', 'num_medica
               'age|number_diagnoses', 'change|num_medications', 'number_diagnoses|time_in_hospital',
               'num_medications|numchange', 'level1_diag1_1', 'level1_diag1_2', 'level1_diag1_3', 'leve
               'level1_diag1_5','level1_diag1_6', 'level1_diag1_7', 'level1_diag1_8')
```

```
# Load required libraries
library(smotefamily)  # For SMOTE
library(caret)        # For train-test split and evaluation
```

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

```
##
## Attaching package: 'caret'
```

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

```
X <- df_pd[, feature_set]
y <- df_pd$readmitted

# Combine X and y
data <- data.frame(X, y)

# Train-test split (80/20)
train_index <- createDataPartition(data$y, p = 0.80, list = FALSE)
X_train <- data[train_index, -ncol(data)]
y_train <- data[train_index, ncol(data)]
X_test <- data[-train_index, -ncol(data)]
y_test <- data[-train_index, ncol(data)]

# Original distribution
cat("Original training dataset shape:\n")
```

```
## Original training dataset shape:
```

```
print(table(y_train))
```

```
## y_train
##     0     1
## 42951  4088
```

```
##1) How does handling class imbalance impact model performance
# Apply SMOTE
smote_result <- SMOTE(X = X_train, target = y_train, K = 5)
train_input_new <- smote_result$data[, -ncol(smote_result$data)]
train_output_new <- smote_result$data[, ncol(smote_result$data)]

cat("New training dataset shape after SMOTE:\n")
```

```
## New training dataset shape after SMOTE:
```

```
print(table(train_output_new))
```

```
## train_output_new
##     0     1
## 42951 40880
```

```
# Replace original training data with SMOTE-augmented data
X_train <- train_input_new
y_train <- train_output_new

# Fit Logistic Regression (GLM with binomial family)
logit_model <- glm(y_train ~ ., data = data.frame(X_train, y_train = as.factor(y_train)), family = bino

# Predict probabilities on test set
prob_pred <- predict(logit_model, newdata = X_test, type = "response")

# Convert probabilities to class labels (threshold = 0.5)
logit_pred <- ifelse(prob_pred > 0.5, "1", "0")

# Confusion Matrix
conf_matrix <- confusionMatrix(as.factor(logit_pred), as.factor(y_test))
print(conf_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 7729  553
##          1 3008  468
##
##                Accuracy : 0.6971
##                  95% CI : (0.6887, 0.7054)
##     No Information Rate : 0.9132
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0854
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.7198
##             Specificity : 0.4584
##          Pos Pred Value : 0.9332
##          Neg Pred Value : 0.1346
##              Prevalence : 0.9132
##          Detection Rate : 0.6573
##    Detection Prevalence : 0.7044
##       Balanced Accuracy : 0.5891
##
##        'Positive' Class : 0
##
```

```r
# Extract performance metrics
accuracy_logit <- conf_matrix$overall["Accuracy"]
precision_logit <- conf_matrix$byClass["Pos Pred Value"]
recall_logit <- conf_matrix$byClass["Sensitivity"]
f1_logit <- 2 * (precision_logit * recall_logit) / (precision_logit + recall_logit)

cat("\nAccuracy:", round(accuracy_logit, 4))
```

```
##
## Accuracy: 0.6971
```

```r
cat("\nPrecision:", round(precision_logit, 4))
```

```
##
## Precision: 0.9332
```

```r
cat("\nRecall:", round(recall_logit, 4))
```

```
##
## Recall: 0.7198
```

```r
cat("\nF1 Score:", round(f1_logit, 4), "\n")
```

```
##
## F1 Score: 0.8128
```

3) Do more complex machine learning models, such as decision trees and random forests, outperform simpler models in predicting hospital readmissions?

##Decison Tree

```r
feature_set_no_int <- c('age', 'time_in_hospital', 'num_procedures', 'num_medications', 'number_outpati
                  'number_emergency_log1p', 'number_inpatient_log1p', 'number_diagnoses', 'metformin',
                  'repaglinide', 'nateglinide', 'chlorpropamide', 'glimepiride', 'glipizide',
                  'glyburide', 'pioglitazone', 'rosiglitazone', 'acarbose',
                  'tolazamide', 'insulin', 'glyburide.metformin',
                  'race_AfricanAmerican', 'race_Asian', 'race_Caucasian',
                  'race_Hispanic', 'race_Other', 'gender_1',
                  'admission_type_id_3', 'admission_type_id_5',
                  'discharge_disposition_id_2', 'discharge_disposition_id_7',
                  'discharge_disposition_id_10', 'discharge_disposition_id_18',
                  'admission_source_id_4', 'admission_source_id_7',
                  'admission_source_id_9', 'max_glu_serum_0',
                  'max_glu_serum_1', 'A1Cresult_0', 'A1Cresult_1',
                  'level1_diag1_1',
                  'level1_diag1_2',
                  'level1_diag1_3',
                  'level1_diag1_4',
                  'level1_diag1_5',
                  'level1_diag1_6',
                  'level1_diag1_7',
                  'level1_diag1_8')
```

```r
X <- df_pd[, feature_set_no_int]
y <- df_pd$readmitted
# Install and load required libraries
# install.packages("smotefamily")  # For SMOTE
# install.packages("caret")  # For splitting data
library(smotefamily)  # SMOTE implementation
library(caret)  # For splitting data

# Combine X and y into a data frame
data <- data.frame(X, y)

# Split the dataset into training and testing sets (80% train, 20% test)
train_index <- createDataPartition(data$y, p = 0.80, list = FALSE)

X_train <- data[train_index, -ncol(data)]
y_train <- data[train_index, ncol(data)]
X_test <- data[-train_index, -ncol(data)]
y_test <- data[-train_index, ncol(data)]

# Print the original class distribution in the training set
print("Original training dataset shape:")
```

```
## [1] "Original training dataset shape:"
```

```r
print(table(y_train))  # This gives the frequency of classes in 'y_train'
```

```
## y_train
##     0     1
## 42951  4088
```

```r
# Apply SMOTE to balance the training set using correct parameters
# The correct syntax is to use 'perc.over' and 'perc.under' differently
smote_result <- SMOTE(X = X_train, target = y_train, K = 5)

# Extract the balanced training data
train_input_new <- smote_result$data[, -ncol(smote_result$data)]  # Features
train_output_new <- smote_result$data[, ncol(smote_result$data)]  # Target

# Print the new class distribution in the balanced training set
print("New training dataset shape after SMOTE:")
```

```
## [1] "New training dataset shape after SMOTE:"
```

```r
print(table(train_output_new))  # New class distribution after SMOTE
```

```
## train_output_new
##     0     1
## 42951 40880
```

```r
# Check the dimensions of the training and testing sets
cat("Training set dimensions: ", dim(train_input_new), "\n")
```

```
## Training set dimensions:  83831 48
```

```r
cat("Testing set dimensions: ", dim(X_test), "\n")
```

```
## Testing set dimensions:  11758 48
```

```r
X_train = train_input_new
y_train = train_output_new
```

```r
library(rpart)

# Combine training features and labels
train_data <- data.frame(X_train, y_train = as.factor(y_train))  # Ensure y_train is factor

# Train Decision Tree
dtree <- rpart(y_train ~ ., data = train_data, method = "class",
               control = rpart.control(maxdepth = 28, minsplit = 10, cp = 0))

# Create test data frame
test_data <- data.frame(X_test)

# Make predictions
logit_pred <- predict(dtree, test_data, type = "class")

# Confusion matrix
confusion_matrix <- confusionMatrix(logit_pred, as.factor(y_test))
print(confusion_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 10174   951
##          1   563    70
##
##                Accuracy : 0.8712
##                  95% CI : (0.865, 0.8772)
##     No Information Rate : 0.9132
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0195
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.94756
##             Specificity : 0.06856
##          Pos Pred Value : 0.91452
##          Neg Pred Value : 0.11058
```

```
##              Prevalence : 0.91317
##          Detection Rate : 0.86528
##    Detection Prevalence : 0.94616
##       Balanced Accuracy : 0.50806
##
##        'Positive' Class : 0
##
```

```r
# Print specific metrics
accuracy_dtree <- confusion_matrix$overall["Accuracy"]
precision_dtree <- confusion_matrix$byClass["Pos Pred Value"]
recall_dtree <- confusion_matrix$byClass["Sensitivity"]
f1_dtree <- 2 * (precision_dtree * recall_dtree) / (precision_dtree + recall_dtree)

cat("\nAccuracy:", round(accuracy_dtree, 4))
```

```
##
## Accuracy: 0.8712
```

```r
cat("\nPrecision:", round(precision_dtree, 4))
```

```
##
## Precision: 0.9145
```

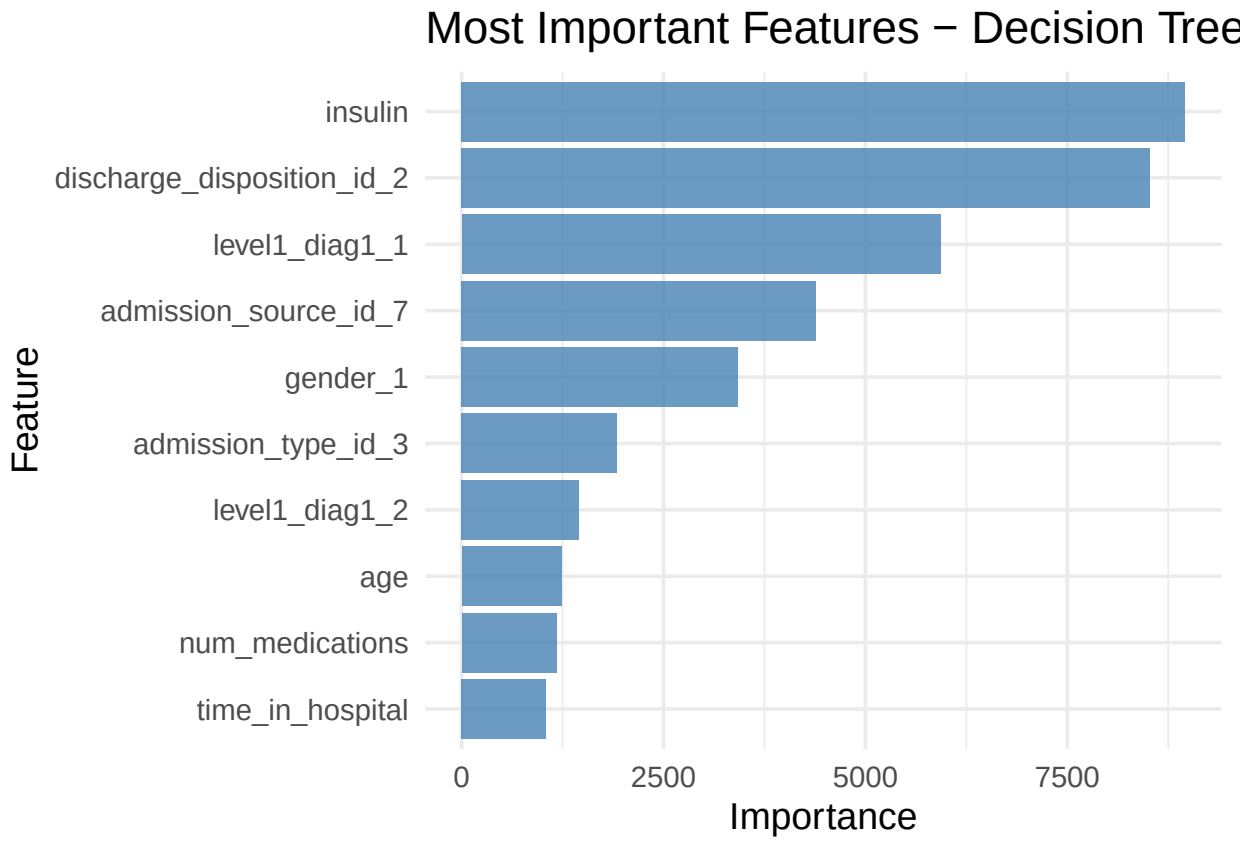
```r
cat("\nRecall:", round(recall_dtree, 4))
```

```
##
## Recall: 0.9476
```

```r
cat("\nF1 Score:", round(f1_dtree, 4), "\n")
```

```
##
## F1 Score: 0.9307
```

```r
# Extract feature importance
importance <- dtree$variable.importance

# Convert to data frame and prepare top 10
importance_df <- data.frame(Feature = names(importance), Importance = importance)
importance_df <- importance_df[order(-importance_df$Importance), ]  # Descending order
top_features <- head(importance_df, 10)

# Reorder for horizontal bar plot
top_features$Feature <- factor(top_features$Feature, levels = rev(top_features$Feature))

# Plot feature importance
ggplot(top_features, aes(x = Feature, y = Importance)) +
  geom_bar(stat = "identity", fill = "steelblue", alpha = 0.8) +
  coord_flip() +
  theme_minimal(base_size = 14) +
  labs(title = "Most Important Features - Decision Tree", x = "Feature", y = "Importance")
```



##random forest

```
X <- df_pd[, feature_set_no_int]
y <- df_pd$readmitted
data <- data.frame(X, y)

# Split the dataset into training and testing sets (80% train, 20% test)
train_index <- createDataPartition(data$y, p = 0.80, list = FALSE)

X_train <- data[train_index, -ncol(data)]
y_train <- data[train_index, ncol(data)]
X_test <- data[-train_index, -ncol(data)]
y_test <- data[-train_index, ncol(data)]

# Print the original class distribution in the training set
print("Original training dataset shape:")
```

```
## [1] "Original training dataset shape:"
```

```
print(table(y_train))  # This gives the frequency of classes in 'y_train'
```

```
## y_train
##     0     1
## 42951  4088
```

```r
# Apply SMOTE to balance the training set using correct parameters
# The correct syntax is to use 'perc.over' and 'perc.under' differently
smote_result <- SMOTE(X = X_train, target = y_train, K = 5)

# Extract the balanced training data
train_input_new <- smote_result$data[, -ncol(smote_result$data)]  # Features
train_output_new <- smote_result$data[, ncol(smote_result$data)]  # Target

# Print the new class distribution in the balanced training set
print("New training dataset shape after SMOTE:")
```

```
## [1] "New training dataset shape after SMOTE:"
```

```r
print(table(train_output_new))  # New class distribution after SMOTE
```

```
## train_output_new
##     0     1
## 42951 40880
```

```r
# Check the dimensions of the training and testing sets
cat("Training set dimensions: ", dim(train_input_new), "\n")
```

```
## Training set dimensions:  83831 48
```

```r
cat("Testing set dimensions: ", dim(X_test), "\n")
```

```
## Testing set dimensions:  11758 48
```

```r
X_train = train_input_new
y_train = train_output_new
```

```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:ggplot2':
##
##     margin
```

```r
library(caret)

# Combine training features and labels
train_data <- data.frame(X_train, y_train = as.factor(y_train))  # Ensure y_train is factor

# Train Random Forest model
rf_model <- randomForest(y_train ~ ., data = train_data, ntree = 100, mtry = sqrt(ncol(X_train)), impor

# Create test data frame
test_data <- data.frame(X_test)

# Make predictions
rf_pred <- predict(rf_model, test_data)

# Confusion matrix
confusion_matrix <- confusionMatrix(rf_pred, as.factor(y_test))
print(confusion_matrix)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 10735  1021
##          1     2     0
##
##                Accuracy : 0.913
##                  95% CI : (0.9078, 0.918)
##     No Information Rate : 0.9132
##     P-Value [Acc > NIR] : 0.5344
##
##                   Kappa : -3e-04
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9998
##             Specificity : 0.0000
##          Pos Pred Value : 0.9132
##          Neg Pred Value : 0.0000
##              Prevalence : 0.9132
##          Detection Rate : 0.9130
##    Detection Prevalence : 0.9998
##       Balanced Accuracy : 0.4999
##
##        'Positive' Class : 0
##
```

```r
# Print specific metrics
accuracy_rm <- confusion_matrix$overall["Accuracy"]
precision_rm <- confusion_matrix$byClass["Pos Pred Value"]
recall_rm <- confusion_matrix$byClass["Sensitivity"]
f1_rm <- 2 * (precision_rm * recall_rm) / (precision_rm + recall_rm)

cat("\nAccuracy:", round(accuracy_rm, 4))
```

```
##
## Accuracy: 0.913
```

```
cat("\nPrecision:", round(precision_rm, 4))
```

```
##
## Precision: 0.9132
```

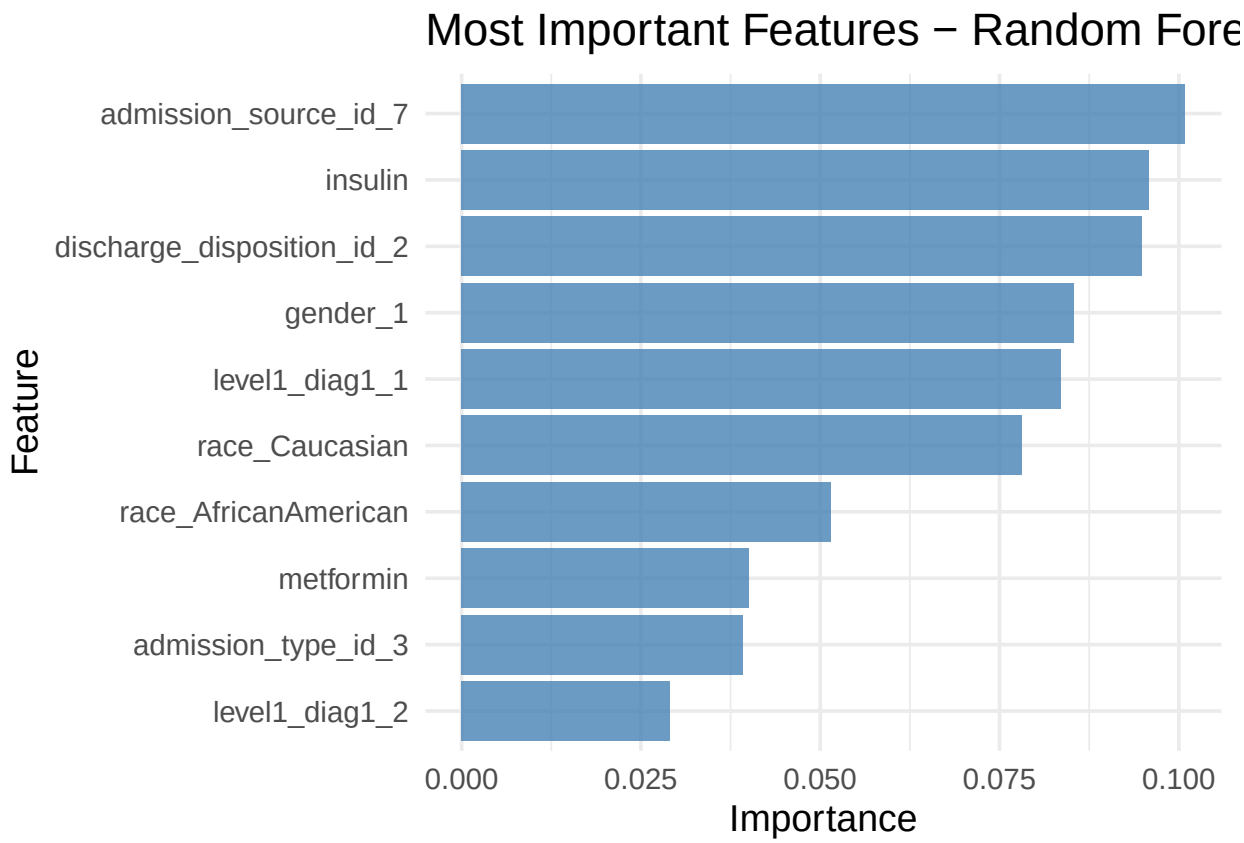
```
cat("\nRecall:", round(recall_rm, 4))
```

```
##
## Recall: 0.9998
```

```
cat("\nF1 Score:", round(f1_rm, 4), "\n")
```

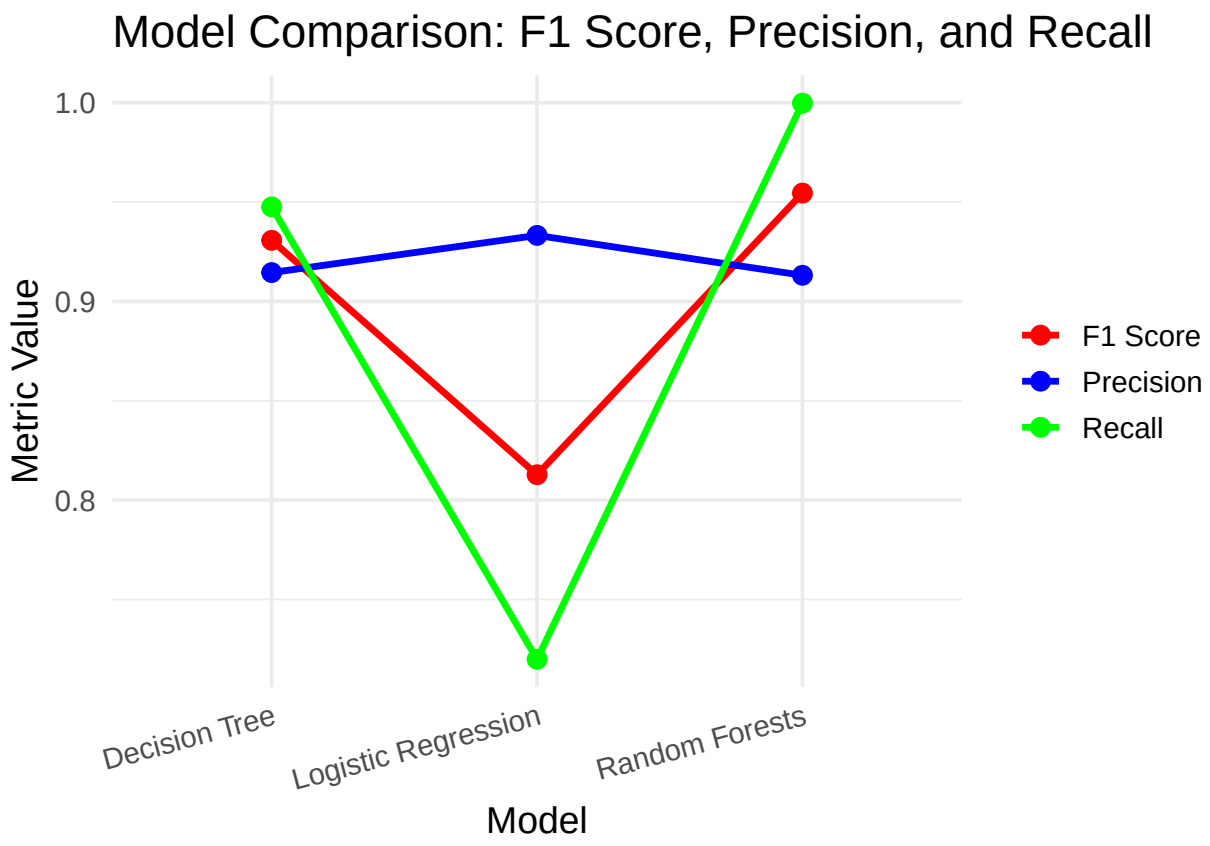
```
##
## F1 Score: 0.9545
```

```
library(randomForest)
library(ggplot2)

# Extract feature importance from Random Forest model
importance_rf <- rf_model$importance[, 1]  # First column corresponds to MeanDecreaseGini

# Convert to data frame and prepare top 10
importance_rf_df <- data.frame(Feature = names(importance_rf), Importance = importance_rf)
importance_rf_df <- importance_rf_df[order(-importance_rf_df$Importance), ]  # Descending order
top_rf_features <- head(importance_rf_df, 10)

# Reorder for horizontal bar plot
top_rf_features$Feature <- factor(top_rf_features$Feature, levels = rev(top_rf_features$Feature))

# Plot feature importance
ggplot(top_rf_features, aes(x = Feature, y = Importance)) +
  geom_bar(stat = "identity", fill = "steelblue", alpha = 0.8) +
  coord_flip() +
  theme_minimal(base_size = 14) +
  labs(title = "Most Important Features - Random Forest", x = "Feature", y = "Importance")
```




```r
# Convert named vectors to numeric
f1_scores <- as.numeric(c(f1_logit, f1_dtree, f1_rm))
precision <- as.numeric(c(precision_logit, precision_dtree, precision_rm))
recall    <- as.numeric(c(recall_logit, recall_dtree, recall_rm))

# Models and metrics
models <- c("Logistic Regression", "Decision Tree", "Random Forests")
metrics <- c("F1 Score", "Precision", "Recall")

# Create data frame manually to avoid misalignment
plot_data <- data.frame(
  Model = rep(models, times = 3),
  Metric = rep(metrics, each = 3),
  Value = c(f1_scores, precision, recall)
)
library(ggplot2)

ggplot(plot_data, aes(x = Model, y = Value, group = Metric, color = Metric)) +
  geom_line(linewidth = 1.2) +
  geom_point(size = 3) +
  theme_minimal(base_size = 14) +
  labs(
    title = "Model Comparison: F1 Score, Precision, and Recall",
    x = "Model",
    y = "Metric Value"
  ) +
```

```
scale_color_manual(values = c("red", "blue", "green")) +
theme(
  axis.text.x = element_text(angle = 15, hjust = 1),
  legend.title = element_blank()
)
```



Conclusion:

In this project, we developed a predictive model to identify the likelihood of hospital readmission among diabetic patients.

The dataset underwent thorough preprocessing involving feature encoding, handling of missing values, standardization, and transformation of skewed variables.

We engineered new features such as service_utilization, numchange, and several interaction terms to capture deeper relationships within the data.

To address the inherent class imbalance in the target variable, we employed the Synthetic Minority Oversampling Technique (SMOTE), ensuring more balanced model training.

We trained and evaluated multiple machine learning models — Logistic Regression, Decision Tree, and Random Forest. Among these, Random Forest performed the best with an F1-score of 0.9545,

demonstrating its effectiveness in handling imbalanced classification and capturing complex patterns in the dataset.

To interpret model predictions, we extracted feature importances using Decision Tree and Random Forest classifiers.

The most influential features contributing to readmission included:

- Time in Hospital

- Discharge Disposition ID

- Number of Diagnoses

- Number of Medications

- Number of Procedures

- Level 1 Diagnosis Code

- Age