ADS503_Final_diabetes | Team 4

Jeffrey Joyner, S M Sultan Mahmud Rahat, Vicky van der Wagt, UE Wang

6/7/2023

Diabetes Hospital Readmission

The objective of this analysis is to predict whether diabetic patients will be readmitted into the hospital within 30 days, within a period longer than 30 days, or not at all, at least not on record.

Data PreProcessing

Import all required packages & libraries

Import dataset Speicying that *na.strings = ?*. While there were no intial null values, upon further examination it was observed that the researchers used "?" to indicate missing values. Specifying the "?" as null for downstream analysis.

```
df <- read.csv("/Users/smsultanmahmudrahat/Library/Mobile Documents/com~apple~CloudDocs/ADS_masters/Data
               na.strings = "?",
               strip.white = TRUE)
#head(df)
dim(df)
```

```
## [1] 101766     50
```

Dataset has 101,766 rows and 50 columns.

```
#describe(df)
```

Examine the unique values in each column There are 2 columns, citoglipvon and examide, that only have one unique value. Many columns also have a very high proportion of observations in one category, and a low number in the other. However,these are all categorical and binary.

Create table to view index position and name of each column This ensures that these features are treated as categorical during analysis.

```
column_table <- data.frame(
  Index = 1:ncol(df),
  Column_Name = colnames(df)
)
```

Used the output of column_table to reference which column indexes need to be converted to factors in the next chunk.

```
get_binary_column_names <- function(df) {
  binary_columns <- sapply(df, function(x) is.factor(x) && nlevels(x) == 2)
  names(df)[binary_columns]
}

binary_column_names <- get_binary_column_names(df)


df$acetohexamide <- ifelse(df$acetohexamide == 'No', 0, 1)
df$tolbutamide <- ifelse(df$tolbutamide == 'No', 0, 1)
df$glipizide.metformin <- ifelse(df$glipizide.metformin == 'No', 0, 1)
df$metformin.rosiglitazone <- ifelse(df$metformin.rosiglitazone == 'No', 0, 1)
df$change <- ifelse(df$change == 'No', 0, 1)
df$diabetesMed <- ifelse(df$diabetesMed == 'No', 0, 1)
```

Convert binary columns to 0 or 1

```
#column_table
to_factor <- c(3:9,11,12, 19:21, 23:50)
df[,to_factor] <- lapply(df[,to_factor] , factor)

#Ensure resepective columns have changed to factor datatype
```

Convert categorical data into factors

```
sum(is.na(df))
```

Check for total null values, and null values by column

```
## [1] 192849
```

```
#create table containing column names with # of missing variables, sorted in decscending order
na <- colSums(is.na(df)) %>% as.data.frame()
na <- cbind(ColName = rownames(na), na)
colnames(na)[2] = "NumNA"
```

```r
na_sort <- na[order(-na$NumNA),c(1,2)]

#attach column containing percentage of missing values
na_sort$PercNA <- round((na_sort$NumNA/(nrow(df))*100),2)
na_sort_cols <- subset(na_sort, select = c(ColName, NumNA, PercNA))

head(na_sort_cols, 10)
```

```
##                               ColName NumNA PercNA
## weight                         weight 98569  96.86
## medical_specialty medical_specialty 49949  49.08
## payer_code                 payer_code 40256  39.56
## race                             race  2273   2.23
## diag_3                         diag_3  1423   1.40
## diag_2                         diag_2   358   0.35
## diag_1                         diag_1    21   0.02
## encounter_id             encounter_id     0   0.00
## patient_nbr               patient_nbr     0   0.00
## gender                         gender     0   0.00
```

There are initially 192,849 null values in this data set. When examining individual columns and associated missing values, it becomes apparent that all the missing values come from 7 columns. These columns and the relative percentage of datapoints missing are weight(96.86%), medical_specialty (49.08%), payer_code (39.56%), race(2.23%), diag_3(1.40%), diag_2(.35%), and diag_1(.02%).

Assess all the missing columns

- weight: since so much of the data is missing and there are no other columns that can be used to impute weight, we will remove this column.
- medical_specialty: TBD based on EDA
- payer_code: TBD based on EDA
- race: just leave as "Unknown"?
- diag_3, diag_2, and diag_1: these are all categorical. they stand for the primary diagnosis, which could be important in analysis. TBD based on EDA

```r
df <- subset(df, select = -c(weight))
```

Ensure there are no duplicate encounters listed in the dataset Encounter_id is the unique identifier of each encounter, so we don't want duplicates. Although patient number is also an identifier, it is acceptable to have duplicates of these, as the observations are individual visits, not patients.

```r
encounter_occ <- data.frame(table(df$encounter_id))
encounter_occ[encounter_occ$Freq > 1,]
```

```
## [1] Var1 Freq
## <0 rows> (or 0-length row.names)
```

There are no duplicate encounters in the diabetes dataset, so no rows need to be removed.

Removing unique identifiers The unique identifiers, encounter_id and patient number have no useful information to modeling. Duplicate patient visits are already captured by "num inpatient visits," so it is redundant.

```
df <- subset(df, select = -c(encounter_id, patient_nbr))
```

```
cont_cols <- sapply(df,is.numeric) #subsetting continuous columns
describe(cont_cols)
```

Subset continuous columns

```
## cont_cols
##        n  missing distinct
##       47        0        2
##
## Value      FALSE  TRUE
## Frequency     39     8
## Proportion  0.83  0.17
```

There are 8 continuous columns.

Check for degenerate columns or colums with one level Only checking for continuous variables, as these are the columns we will scale; we want to ensure that scaling these will not induce any NAs.

```
degeneratecols <- nearZeroVar(cont_cols)
degeneratecols
```

```
## integer(0)
```

```
get_univariate_colnames <- function(df) {
  univariate_columns <- sapply(df, function(x) is.factor(x) && nlevels(x) == 1)
  names(df)[univariate_columns]
}
univariate_column_names <- get_univariate_colnames(df)
univariate_column_names
```

```
## [1] "examide"     "citoglipton"
```

```
#remove univariate columns
df <- subset(df, select = -c(citoglipton, examide))
```

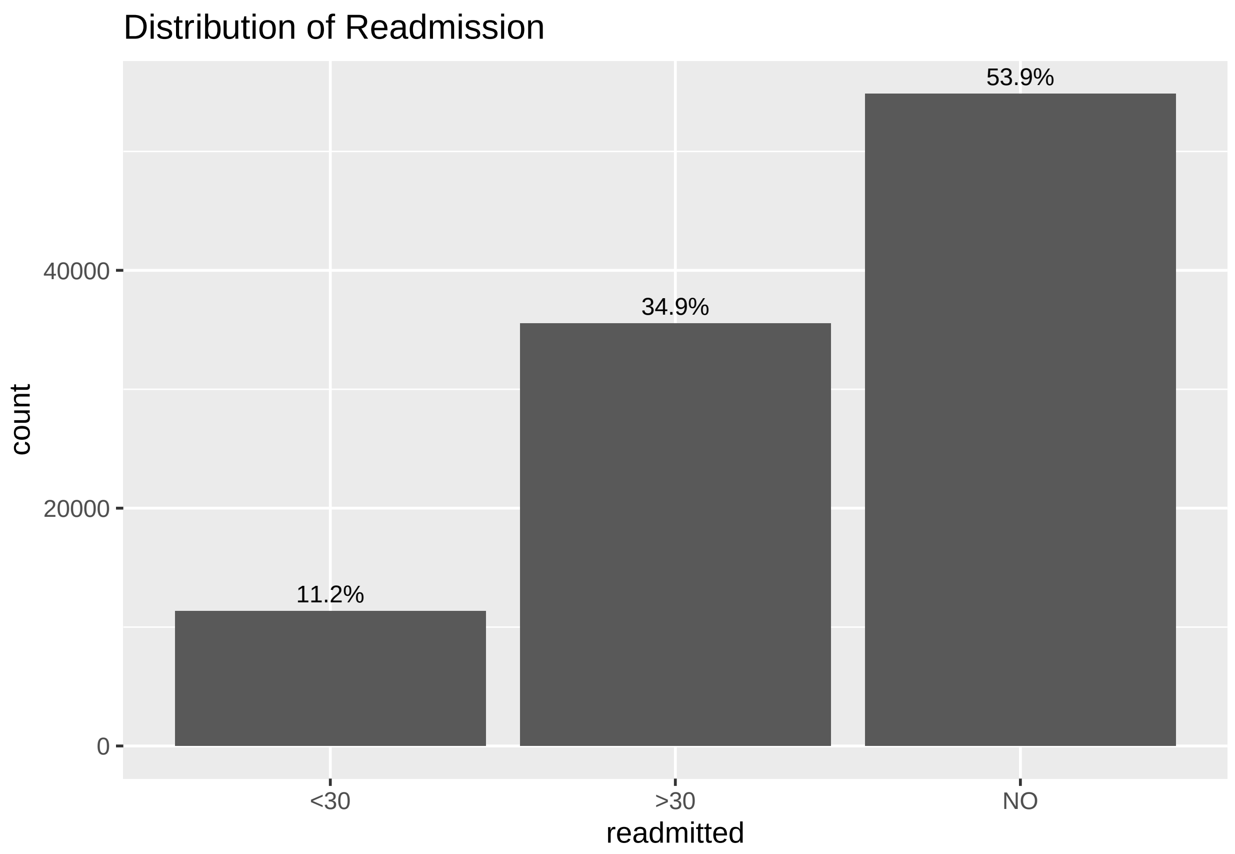
There are no degenerate columns.

Scale continuous predictors Only scaling numeric columns, as the categorical variables(stored as factors) don't have numerical meaning/magnitude.

```
cont_index <- which(cont_cols)

scaled_data <- preProcess(df[, cont_index], method = c("center", "scale"))
scaled_df <- predict(scaled_data, df[, cont_index])
#replacing the continuous features with their scaled values in the original df
df[, cont_index] <- scaled_df
```

Exploratory Data Analysis (EDA)

```
library(ggplot2)
# Create a count plot
ggplot(df, aes(x = readmitted)) + geom_bar() +
  labs(title = "Distribution of Readmission") +
  geom_text(
            aes(label = paste0(round((..count..)/sum(..count..) * 100, 1), "%")),
            stat = "count", vjust = -0.5, size = 3)
```



To convert the target variable into a binary variable, we convert the not readmitted and > 30 readmitted into 0, and readmitted within 30 days as 1, to better analyze our data.

```
# Convert 'NO' and '>30' to 0, and '<30' to 1
df$readmitted <- ifelse(df$readmitted == 'NO' | df$readmitted == '>30', 0, 1)
table(df$readmitted)
```

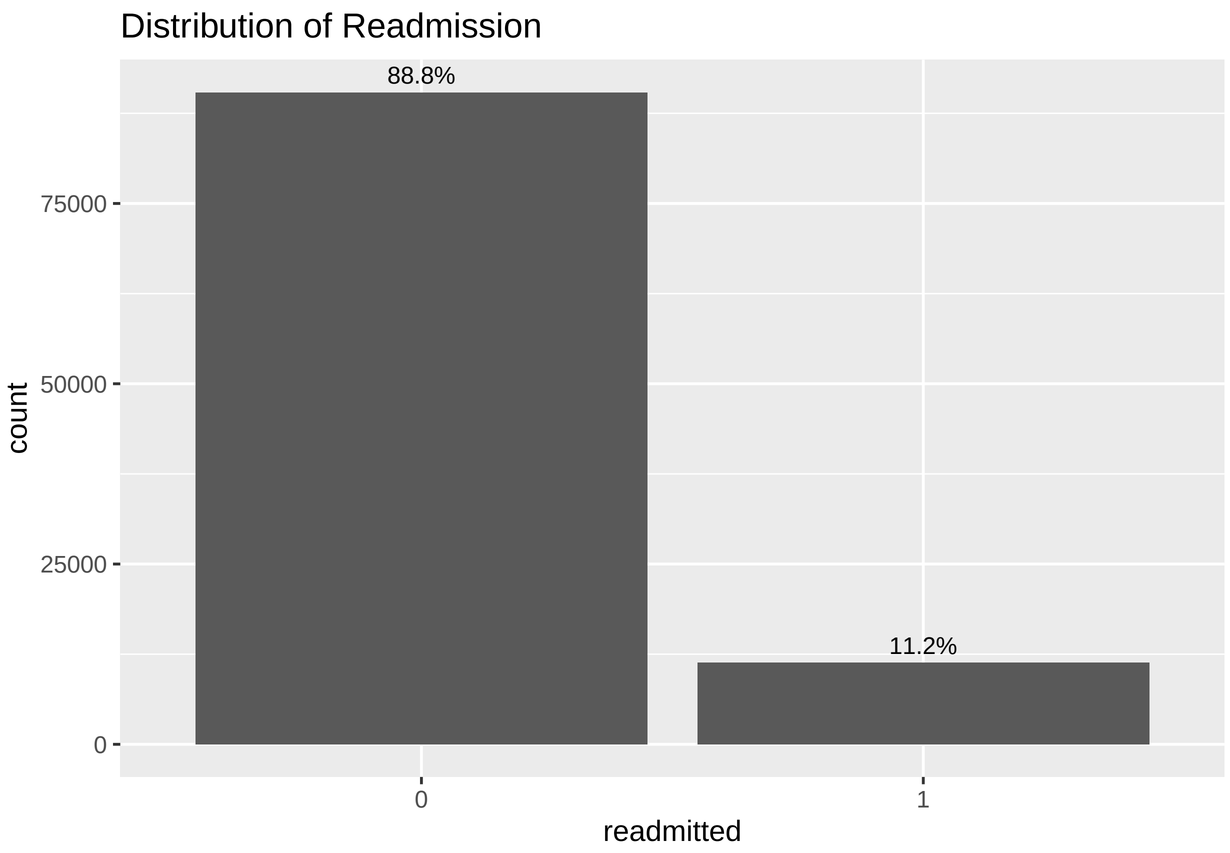
```
##
```

```
##     0     1
## 90409 11357
```

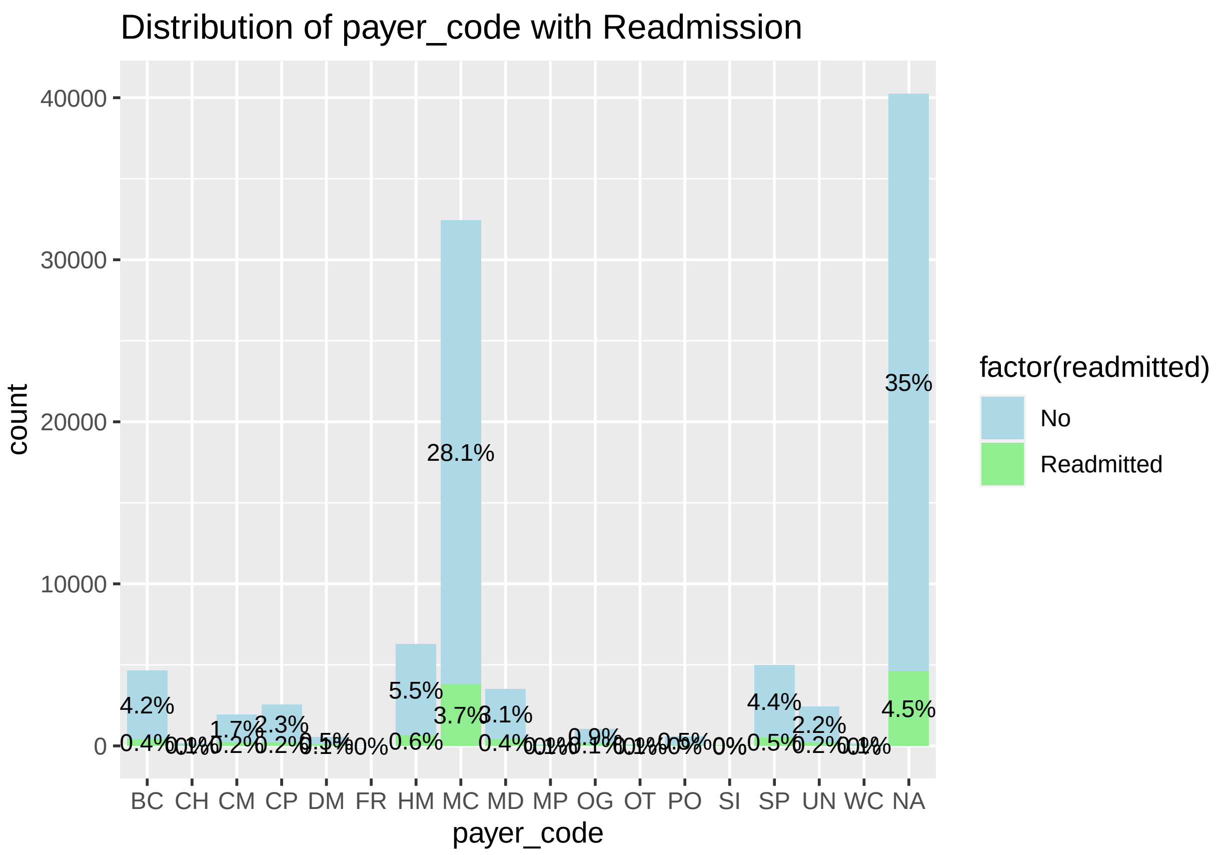
```
df$readmitted <- as.factor(df$readmitted)
```

Uneven class distribution

```
# replot the bar plot
ggplot(df, aes(x = readmitted)) + geom_bar() +
  labs(title = "Distribution of Readmission") +
  geom_text(
           aes(label = paste0(round((..count..)/sum(..count..) * 100, 1), "%")),
           stat = "count", vjust = -0.5, size = 3)
```



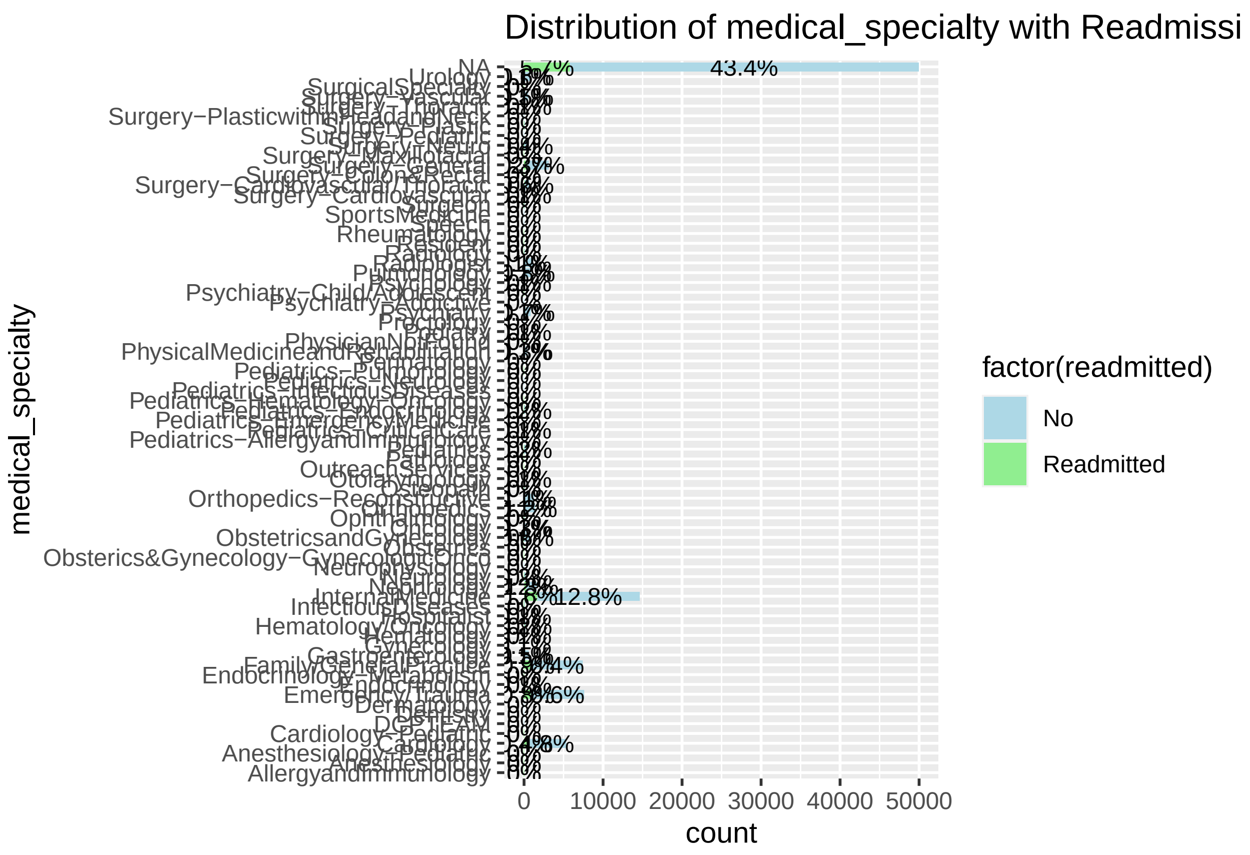
```
# Create a stacked bar plot with 'payer_code' and 'readmitted' distribution
ggplot(df, aes(x = payer_code, fill = factor(readmitted))) + geom_bar() +
  labs(title = "Distribution of payer_code with Readmission") +
  scale_fill_manual(values = c("0" = "lightblue", "1" = "lightgreen"),
                    labels = c("0" = "No", "1" = "Readmitted")) +
  geom_text(
    aes(label = paste0(round((..count..)/sum(..count..) * 100, 1), "%")),
    stat = "count", position = position_stack(vjust = 0.5), size = 3 )
```



After plotting the distribution of 'payer_code' with the readmission status, it was observed that approximately 40% of the 'payer_code' values were missing. Furthermore, the 'payer_code' does not appear to be significant impacted on whether a patient is readmitted or not.

medical_specialty

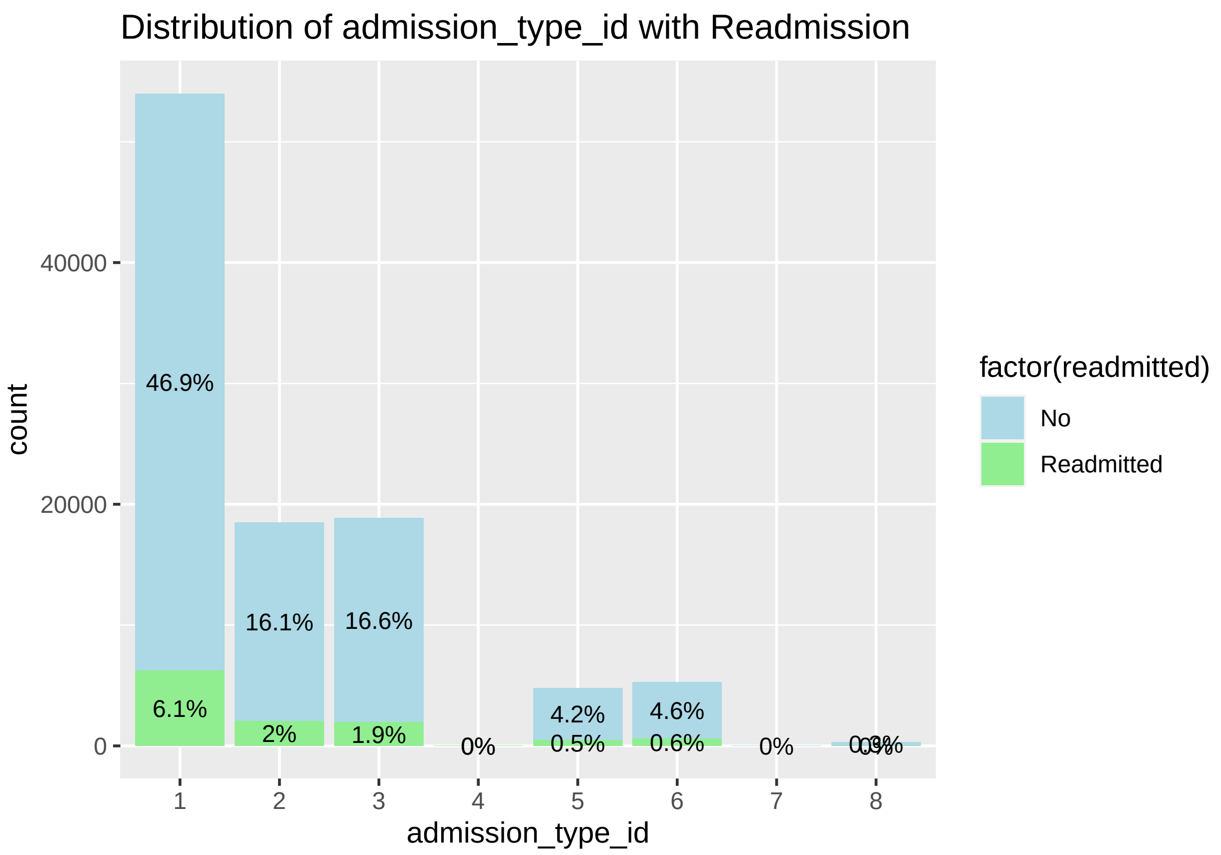
```
# Create a bar plot with the sorted and filtered data
ggplot(df, aes(x = medical_specialty, fill = factor(readmitted))) + geom_bar() +
  labs(title = "Distribution of medical_specialty with Readmission") +
  scale_fill_manual(values = c("0" = "lightblue", "1" = "lightgreen"),
                    labels = c("0" = "No", "1" = "Readmitted")) +
  geom_text(
    aes(label = paste0(round((..count..)/sum(..count..) * 100, 1), "%")),
    stat = "count", position = position_stack(vjust = 0.5), size = 3) +  coord_flip()
```



After plotting the distribution of 'medical_specialty' with the readmission status, it was observed that approximately 50% of the 'medical_specialty' values were missing. Furthermore, the 'medical_specialty' does not appear to be significant impacted on whether a patient is readmitted or not.

admission_type_id

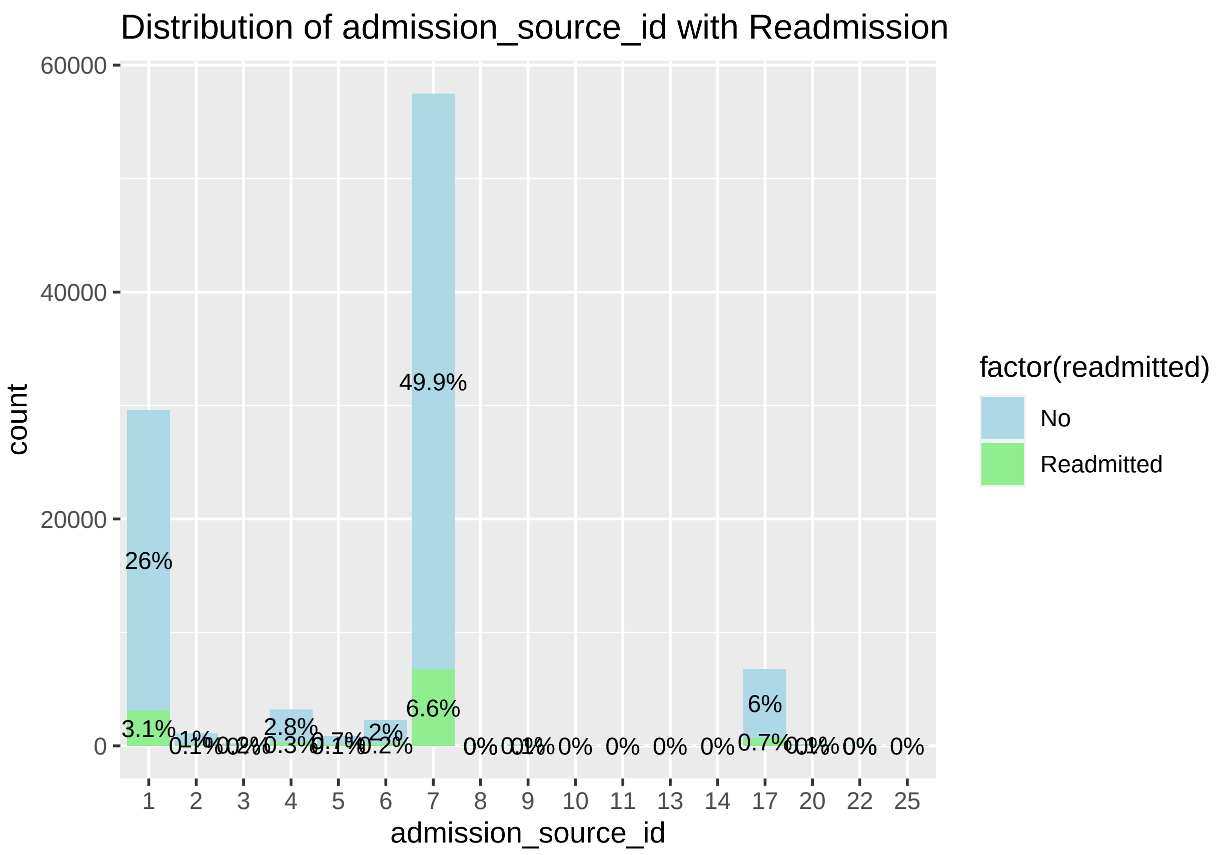
```r
# Create a stacked bar plot with 'admission_type_id' and 'readmitted' distribution
ggplot(df, aes(x = admission_type_id, fill = factor(readmitted))) + geom_bar() +
  labs(title = "Distribution of admission_type_id with Readmission") +
  scale_fill_manual(values = c("0" = "lightblue", "1" = "lightgreen"),
                    labels = c("0" = "No", "1" = "Readmitted")) +
  geom_text(
    aes(label = paste0(round((..count..)/sum(..count..) * 100, 1), "%")),
    stat = "count", position = position_stack(vjust = 0.5), size = 3 )
```



There are 53.1% of patients in the emergency admission type, and 18.2% are classified as urgent admission types. 18.5% are labeled as elective admission types. “Not Available” and “Null” each have 4.7% and 5.2% of the total adjustment type. Patients readmission rates in emergency admission types are higher than in other admission types.

admission_source_id

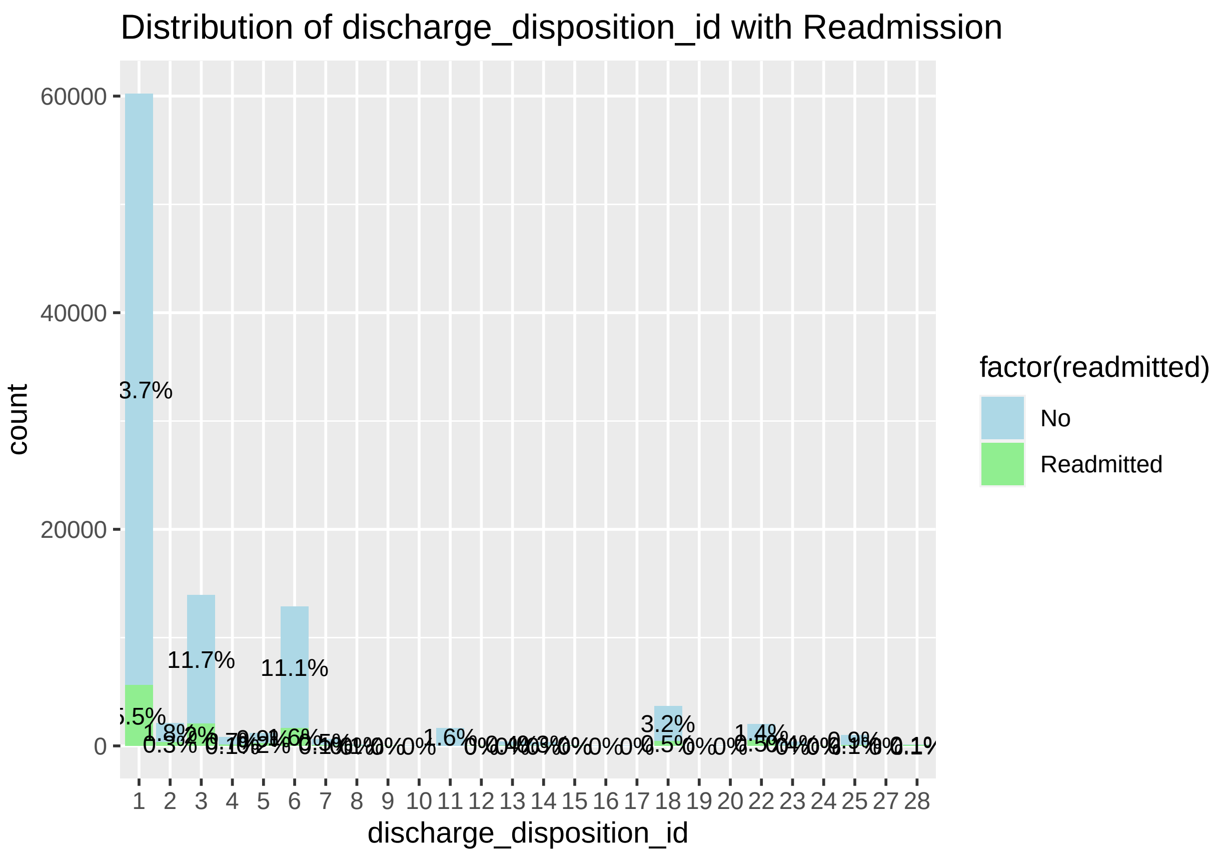
```
# Create a stacked bar plot with 'admission_source_id' and 'readmitted' distribution
ggplot(df, aes(x = admission_source_id, fill = factor(readmitted))) + geom_bar() +
  labs(title = "Distribution of admission_source_id with Readmission") +
  scale_fill_manual(values = c("0" = "lightblue", "1" = "lightgreen"),
                    labels = c("0" = "No", "1" = "Readmitted")) +
  geom_text(
    aes(label = paste0(round((..count..)/sum(..count..) * 100, 1), "%")),
    stat = "count", position = position_stack(vjust = 0.5), size = 3 )
```



We can tell from the above plot that 56.5% of patients are from the emergency room, 29.1% are from physician referrals, and 6.7% are transferred from another health agency. 3.1% are transferred from the HMO rederral, and 2.2% are transferred from another health care facility. Patients from emergency room admission sources have a higher rate of readmission than patients from other sources.

discharge_disposition_id

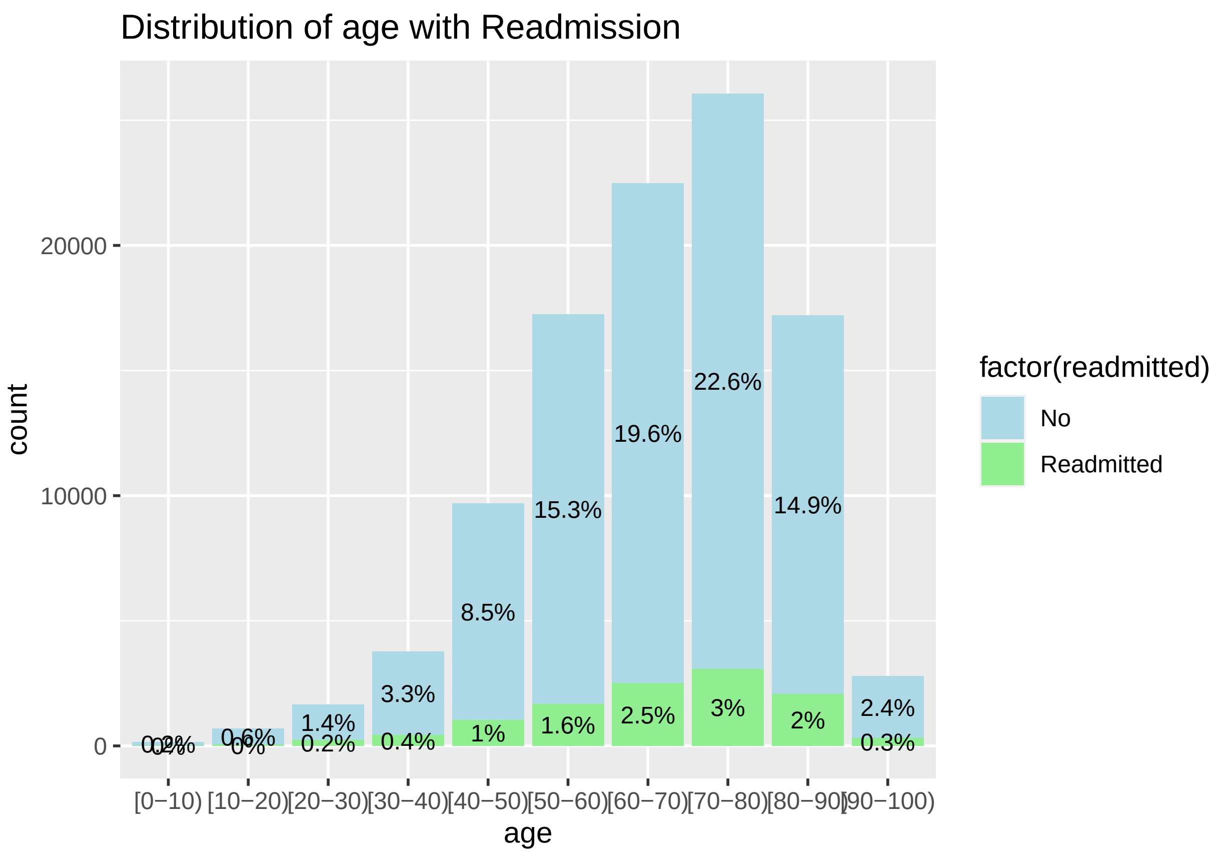
```
# Create a stacked bar plot with 'discharge_disposition_id' and 'readmitted' distribution
ggplot(df, aes(x = discharge_disposition_id, fill = factor(readmitted))) + geom_bar() +
  labs(title = "Distribution of discharge_disposition_id with Readmission") +
  scale_fill_manual(values = c("0" = "lightblue", "1" = "lightgreen"),
                    labels = c("0" = "No", "1" = "Readmitted")) +
  geom_text(
    aes(label = paste0(round((..count..)/sum(..count..) * 100, 1), "%")),
    stat = "count", position = position_stack(vjust = 0.5), size = 3 )
```



59.2% of patients are discharged home; 13.7% are discharged or transferred to another short-term hospital; and 12.7% are discharged or transferred home with home health services. Patients who are discharged home have a 5.5% rate of being readmitted, which is higher than other discharge disposition types.

age

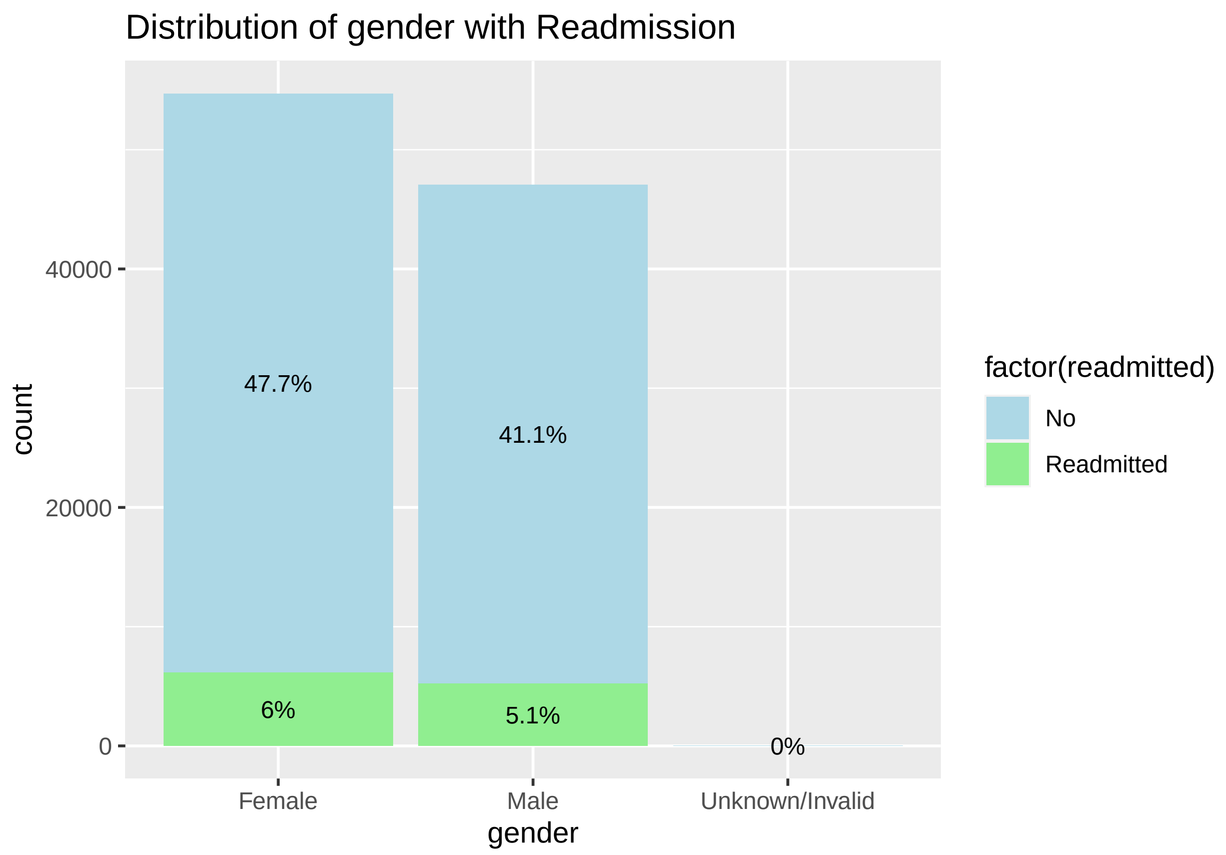
```r
# Create a stacked bar plot with 'age' and 'readmitted' distribution
ggplot(df, aes(x = age, fill = factor(readmitted))) + geom_bar() +
  labs(title = "Distribution of age with Readmission") +
  scale_fill_manual(values = c("0" = "lightblue", "1" = "lightgreen"),
                    labels = c("0" = "No", "1" = "Readmitted")) +
  geom_text(
    aes(label = paste0(round((..count..)/sum(..count..) * 100, 1), "%")),
    stat = "count", position = position_stack(vjust = 0.5), size = 3 )
```



25.6% of diabetic patients are aged between 70 and 80, followed by the age 60–70, which has 22.4%, and the age 80–90, which has 16.9% of the population. Age 70–80 also has the highest readmission rate of any age group.

gender

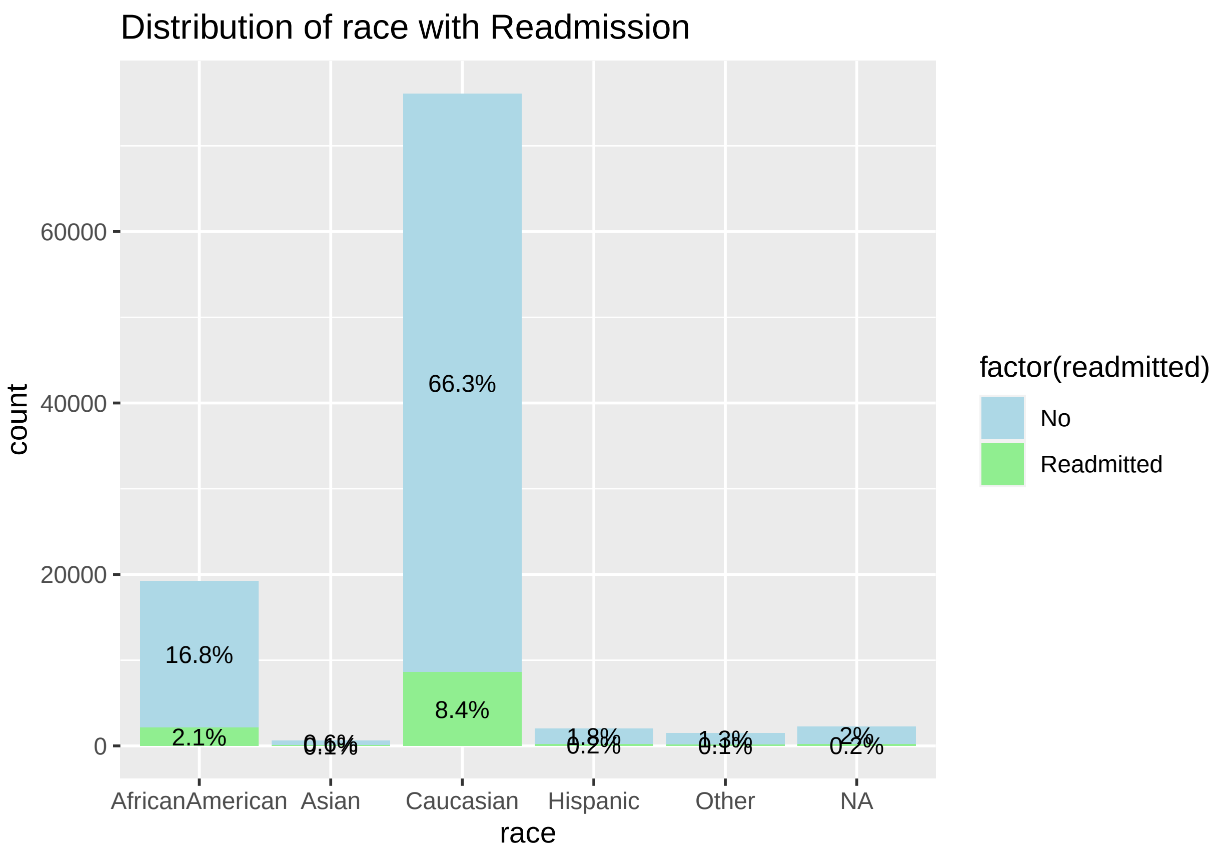
```
# Create a stacked bar plot with 'gender' and 'readmitted' distribution
ggplot(df, aes(x = gender, fill = factor(readmitted))) + geom_bar() +
  labs(title = "Distribution of gender with Readmission") +
  scale_fill_manual(values = c("0" = "lightblue", "1" = "lightgreen"),
                    labels = c("0" = "No", "1" = "Readmitted")) +
  geom_text(
    aes(label = paste0(round((..count..)/sum(..count..) * 100, 1), "%")),
    stat = "count", position = position_stack(vjust = 0.5), size = 3 )
```



Female diabetes patients outnumber male patients by a margin of 47.7%, and their readmission rate is 6% greater than male patients'.

race

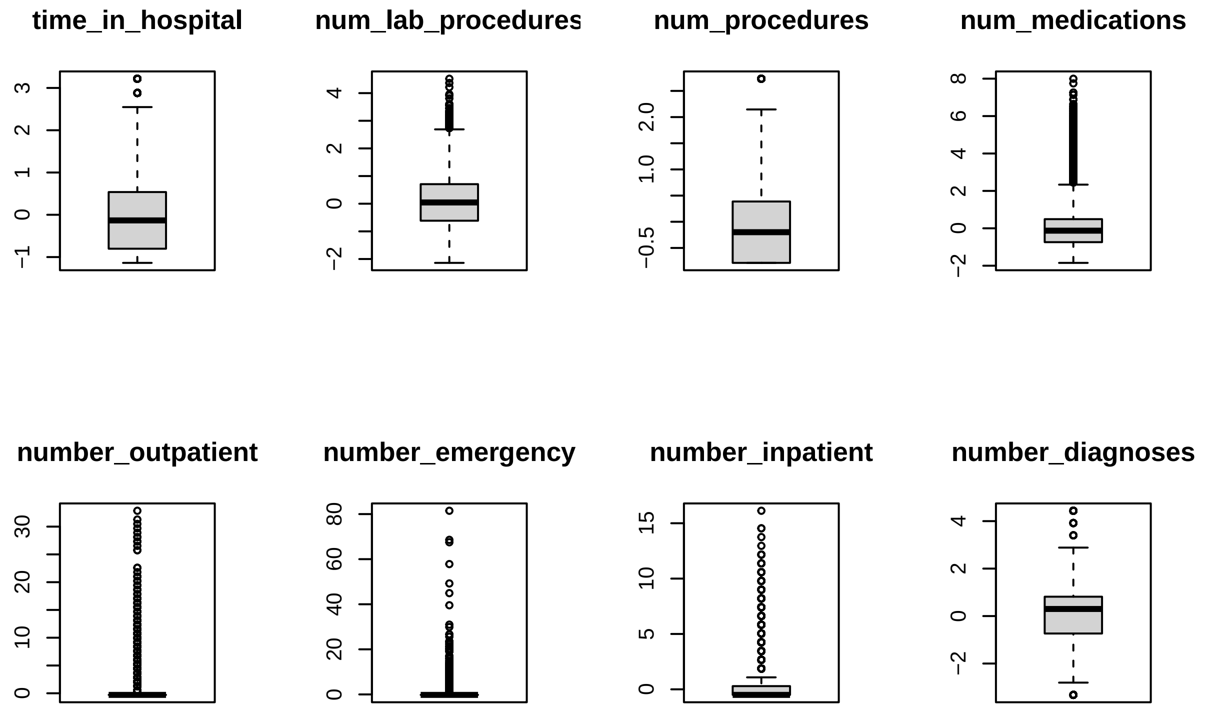
```r
# Create a stacked bar plot with 'race' and 'readmitted' distribution
ggplot(df, aes(x = race, fill = factor(readmitted))) + geom_bar() +
  labs(title = "Distribution of race with Readmission") +
  scale_fill_manual(values = c("0" = "lightblue", "1" = "lightgreen"),
                    labels = c("0" = "No", "1" = "Readmitted")) +
  geom_text(
    aes(label = paste0(round((..count..)/sum(..count..) * 100, 1), "%")),
    stat = "count", position = position_stack(vjust = 0.5), size = 3)
```



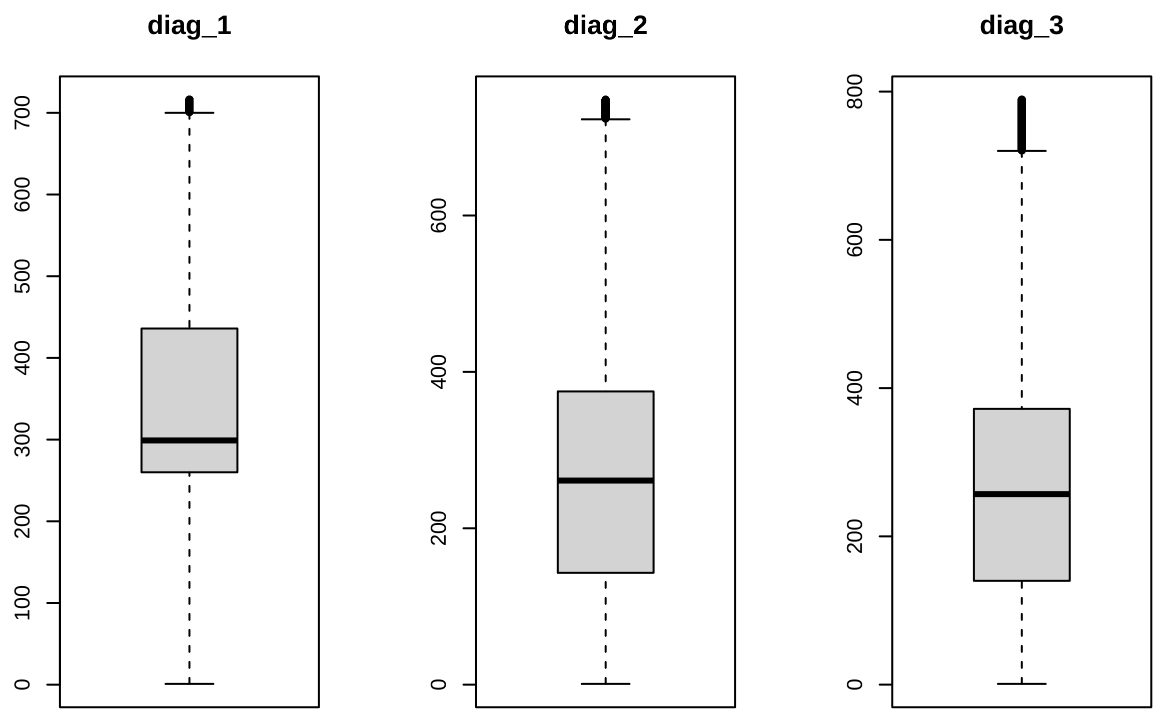
74.7% of the deabetic patients are white, and they also have the highest readmission rate of 8.4% among all race groups.

num_medications

```
par(mfrow = c(2,4))
boxplot(df$time_in_hospital, main = "time_in_hospital")
boxplot(df$num_lab_procedures, main = "num_lab_procedures")
boxplot(df$num_procedures, main = "num_procedures")
boxplot(df$num_medications, main = "num_medications")
boxplot(df$number_outpatient, main = "number_outpatient")
boxplot(df$number_emergency, main = "number_emergency")
boxplot(df$number_inpatient, main = "number_inpatient")
boxplot(df$number_diagnoses, main = "number_diagnoses")
```

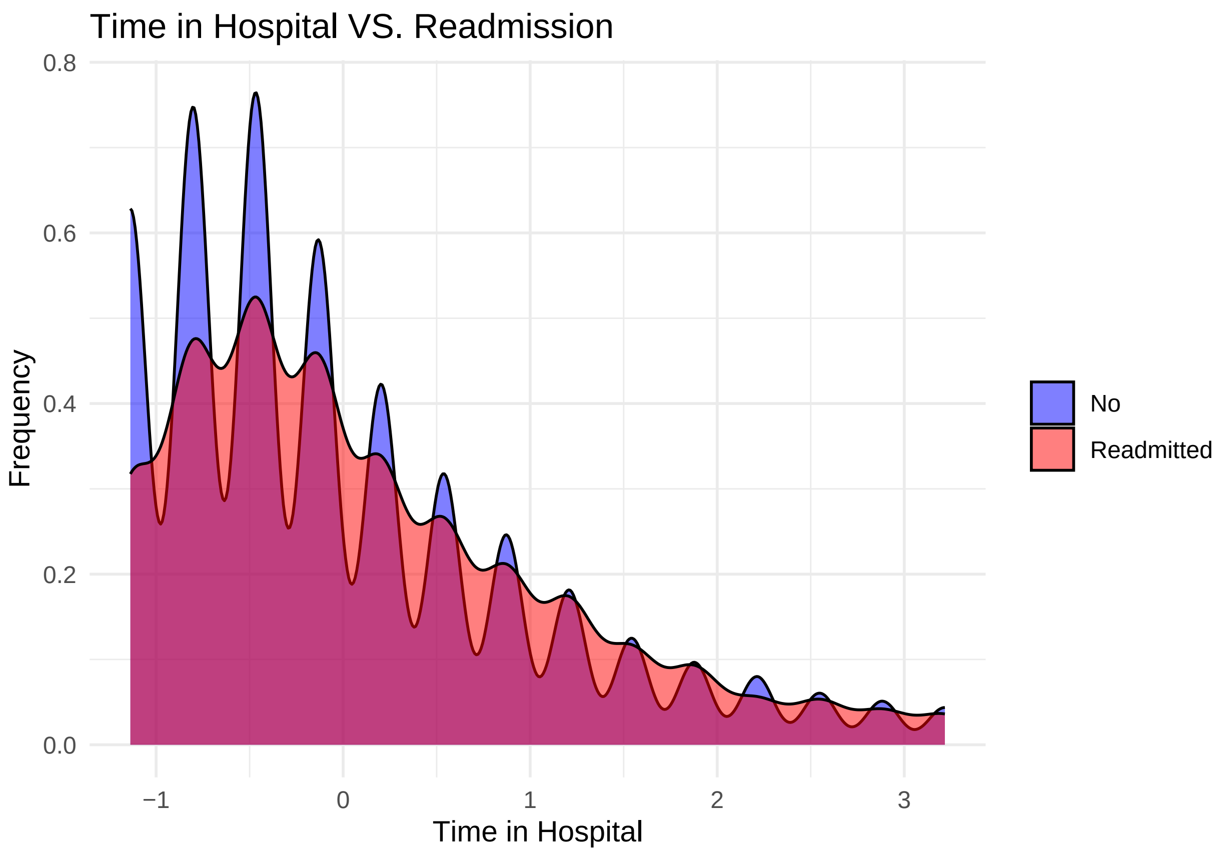


```
par(mfrow = c(1,3))
boxplot(df$diag_1, main = "diag_1")
boxplot(df$diag_2, main = "diag_2")
boxplot(df$diag_3, main = "diag_3")
```

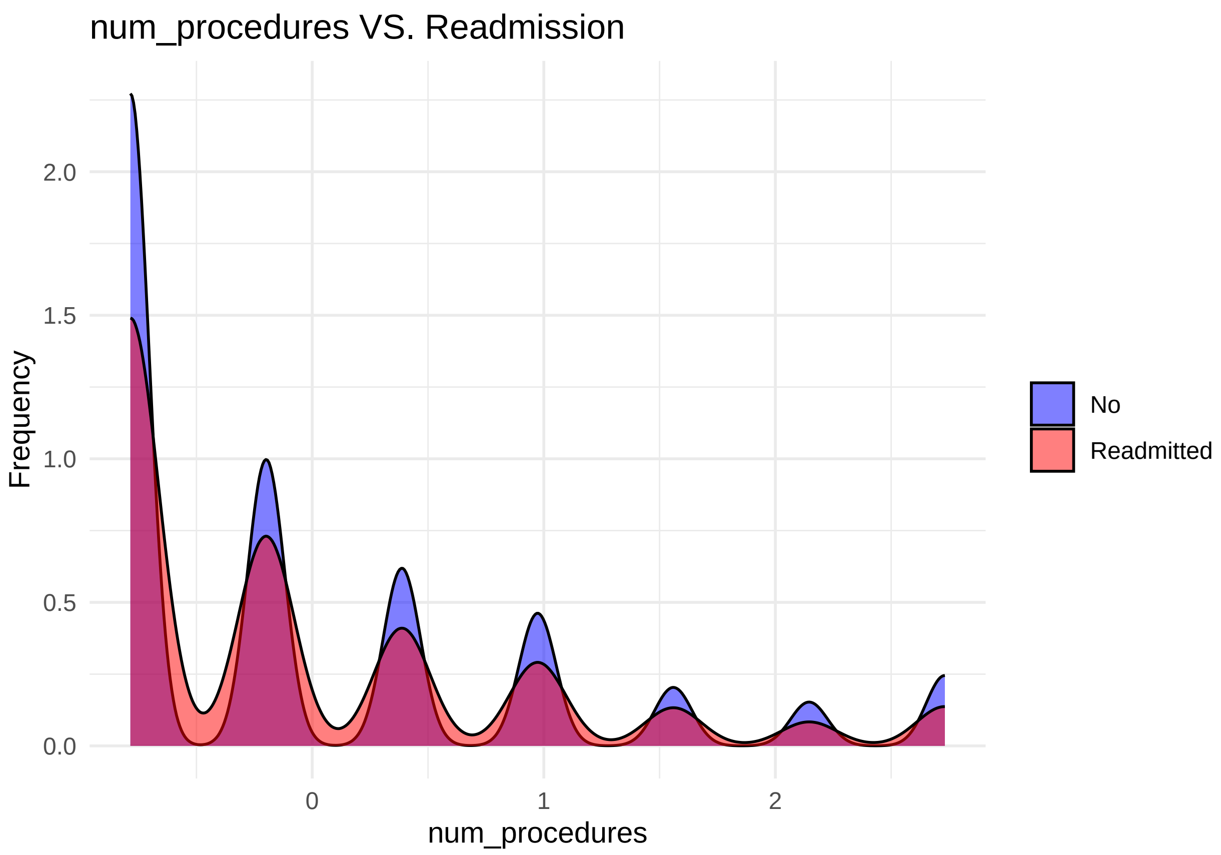


The diag_3 extra diagnostic mean value is substantially lower than the first two diagnoses, which have mean values of about 250 and 420, respectively.

```r
fig <- ggplot(df, aes(x = time_in_hospital, fill = factor(readmitted))) +
  geom_density(alpha = 0.5) +
  scale_fill_manual(values = c("blue", "red"), labels = c("No", "Readmitted")) +
  xlab("Time in Hospital") +
  ylab("Frequency") +
  ggtitle("Time in Hospital VS. Readmission") +
  theme_minimal() +
  theme(legend.title = element_blank())
fig
```



```r
fig <- ggplot(df, aes(x = num_procedures, fill = factor(readmitted))) +
  geom_density(alpha = 0.5) +
  scale_fill_manual(values = c("blue", "red"), labels = c("No", "Readmitted")) +
  xlab("num_procedures") +
  ylab("Frequency") +
  ggtitle("num_procedures VS. Readmission") +
  theme_minimal() +
  theme(legend.title = element_blank())
fig
```



The time in hospital and num_procedures variables show that the readmitted rate is not specifically associated with a certain number; no sign shows that the patient has a higher readmitted chance than other values.

```
df$max_glu_serum <- ifelse(df$max_glu_serum == "None",  0, df$max_glu_serum);
df$max_glu_serum <- ifelse(df$max_glu_serum == "Norm",  100, df$max_glu_serum);
df$max_glu_serum <- ifelse(df$max_glu_serum == ">200",  200, df$max_glu_serum);
df$max_glu_serum <- ifelse(df$max_glu_serum == ">300",  300, df$max_glu_serum);
```

```
table(df$max_glu_serum)
```

```
##
##     0     1     2     4
## 96420  1485  1264  2597
```

```
df$A1Cresult <- ifelse(df$A1Cresult == "None",  0, df$A1Cresult);
df$A1Cresult <- ifelse(df$A1Cresult == "Norm",  5, df$A1Cresult);
df$A1Cresult <- ifelse(df$A1Cresult == ">7",    7, df$A1Cresult);
df$A1Cresult <- ifelse(df$A1Cresult == ">8",    8, df$A1Cresult);
table(df$A1Cresult)
```

```
##
##     0     1     2     4
## 84748  3812  8216  4990
```

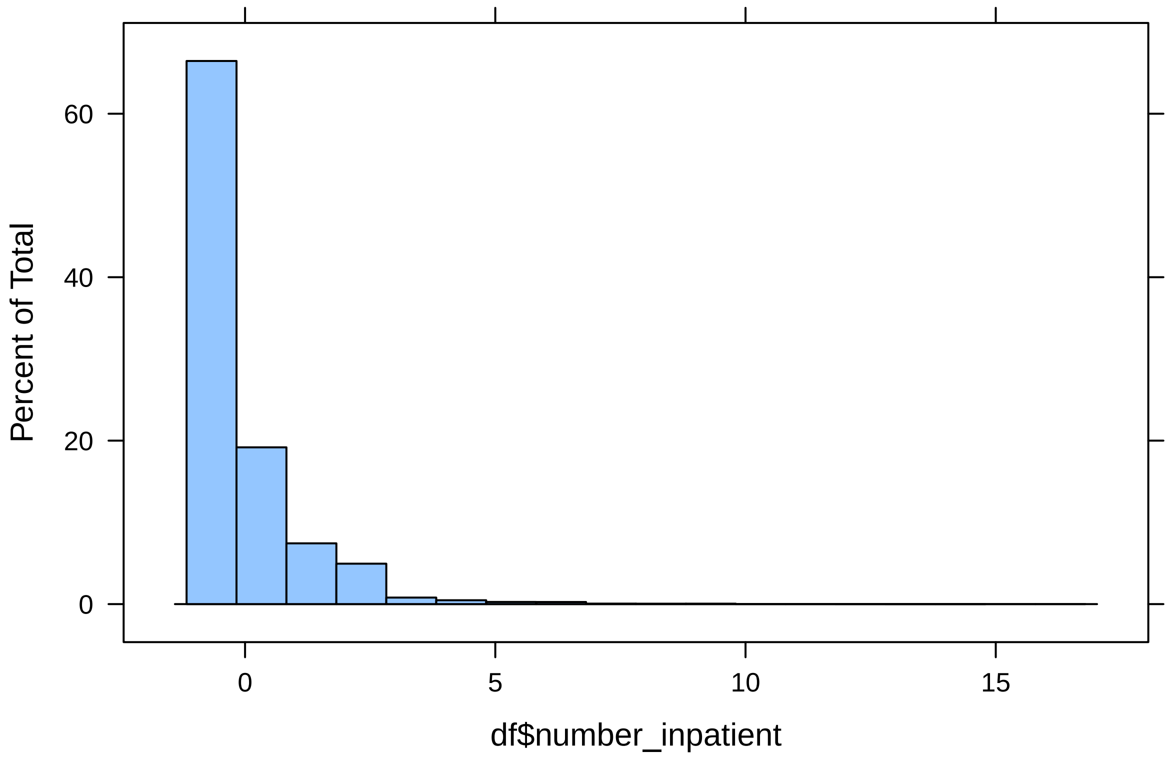
diag_1, diag_2, diag_3, the values are representing different type of medical condition. https://icd.codes/icd9cm (details)

```
`%notin%` <- Negate(`%in%`)

levels(df$diag_1)[levels(df$diag_1) %notin% as.factor(c(390:459, 785, 460:519, 786, 520:579, 787, seq(25
levels(df$diag_1)[levels(df$diag_1) %in% as.factor(c(390:459, 785))] <- "Circulatory"
levels(df$diag_1)[levels(df$diag_1) %in% as.factor(c(460:519, 786))] <- "Respiratory"
levels(df$diag_1)[levels(df$diag_1) %in% as.factor(c(520:579, 787))] <- "Digestive"
levels(df$diag_1)[levels(df$diag_1) %in% as.factor(c(seq(250,250.99, 0.01)))] <- ""
levels(df$diag_1)[levels(df$diag_1) %in% as.factor(c(800:999))] <- "Injury"
levels(df$diag_1)[levels(df$diag_1) %in% as.factor(c(710:739))] <- "Musculoskeletal"
levels(df$diag_1)[levels(df$diag_1) %in% as.factor(c(580:629, 788))] <- "Genitourinary"
levels(df$diag_1)[levels(df$diag_1) %in% as.factor(c(140:239))] <- "Neoplasms"

levels(df$diag_2)[levels(df$diag_2) %notin% as.factor(c(390:459, 785, 460:519, 786, 520:579, 787, seq(25
levels(df$diag_2)[levels(df$diag_2) %in% as.factor(c(390:459, 785))] <- "Circulatory"
levels(df$diag_2)[levels(df$diag_2) %in% as.factor(c(460:519, 786))] <- "Respiratory"
levels(df$diag_2)[levels(df$diag_2) %in% as.factor(c(520:579, 787))] <- "Digestive"
levels(df$diag_2)[levels(df$diag_2) %in% as.factor(c(seq(250,250.99, 0.01)))] <- "Diabetes"
levels(df$diag_2)[levels(df$diag_2) %in% as.factor(c(800:999))] <- "Injury"
levels(df$diag_2)[levels(df$diag_2) %in% as.factor(c(710:739))] <- "Musculoskeletal"
levels(df$diag_2)[levels(df$diag_2) %in% as.factor(c(580:629, 788))] <- "Genitourinary"
levels(df$diag_2)[levels(df$diag_2) %in% as.factor(c(140:239))] <- "Neoplasms"

levels(df$diag_3)[levels(df$diag_3) %notin% as.factor(c(390:459, 785, 460:519, 786, 520:579, 787, seq(25
levels(df$diag_3)[levels(df$diag_3) %in% as.factor(c(390:459, 785))] <- "Circulatory"
levels(df$diag_3)[levels(df$diag_3) %in% as.factor(c(460:519, 786))] <- "Respiratory"
levels(df$diag_3)[levels(df$diag_3) %in% as.factor(c(520:579, 787))] <- "Digestive"
levels(df$diag_3)[levels(df$diag_3) %in% as.factor(c(seq(250,250.99, 0.01)))] <- "Diabetes"
levels(df$diag_3)[levels(df$diag_3) %in% as.factor(c(800:999))] <- "Injury"
levels(df$diag_3)[levels(df$diag_3) %in% as.factor(c(710:739))] <- "Musculoskeletal"
levels(df$diag_3)[levels(df$diag_3) %in% as.factor(c(580:629, 788))] <- "Genitourinary"
levels(df$diag_3)[levels(df$diag_3) %in% as.factor(c(140:239))] <- "Neoplasms"
```

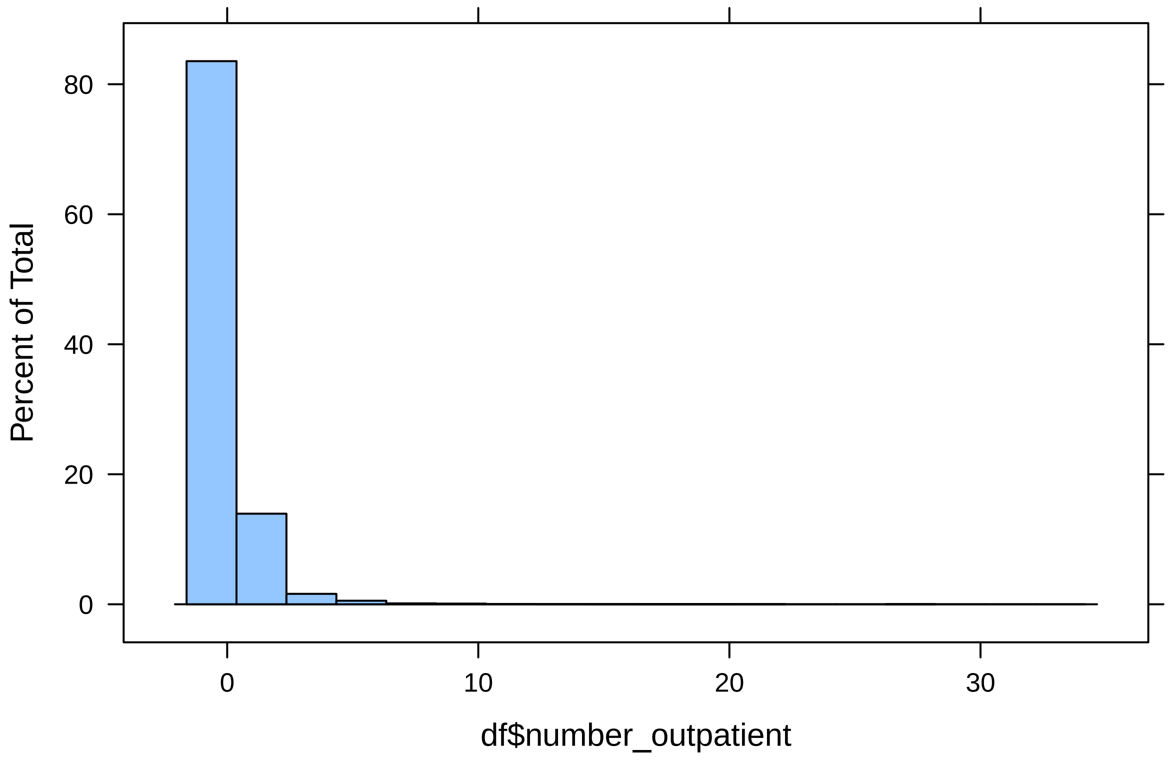
```
## Group the rest of diabetes medications into a new column called num_med and num_changes

keys <- c('metformin', 'repaglinide', 'nateglinide', 'chlorpropamide', 'glimepiride', 'glipizide', 'gly

df$num_med <- 0
df$num_changes <- 0
for(key in keys){
  df$num_med <- ifelse(df[key] != 'No', df$num_med + 1, df$num_med)
  df$num_changes <- ifelse((df[key] == 'Up' | df[key] == 'Down'), df$num_changes + 1, df$num_changes)
}
df <- subset(df, select = -c(metformin, repaglinide, nateglinide, chlorpropamide, glimepiride, glipizid
df$num_med <-factor(df$num_med)
df$num_changes <-factor(df$num_changes)
```

#Normalizing different features

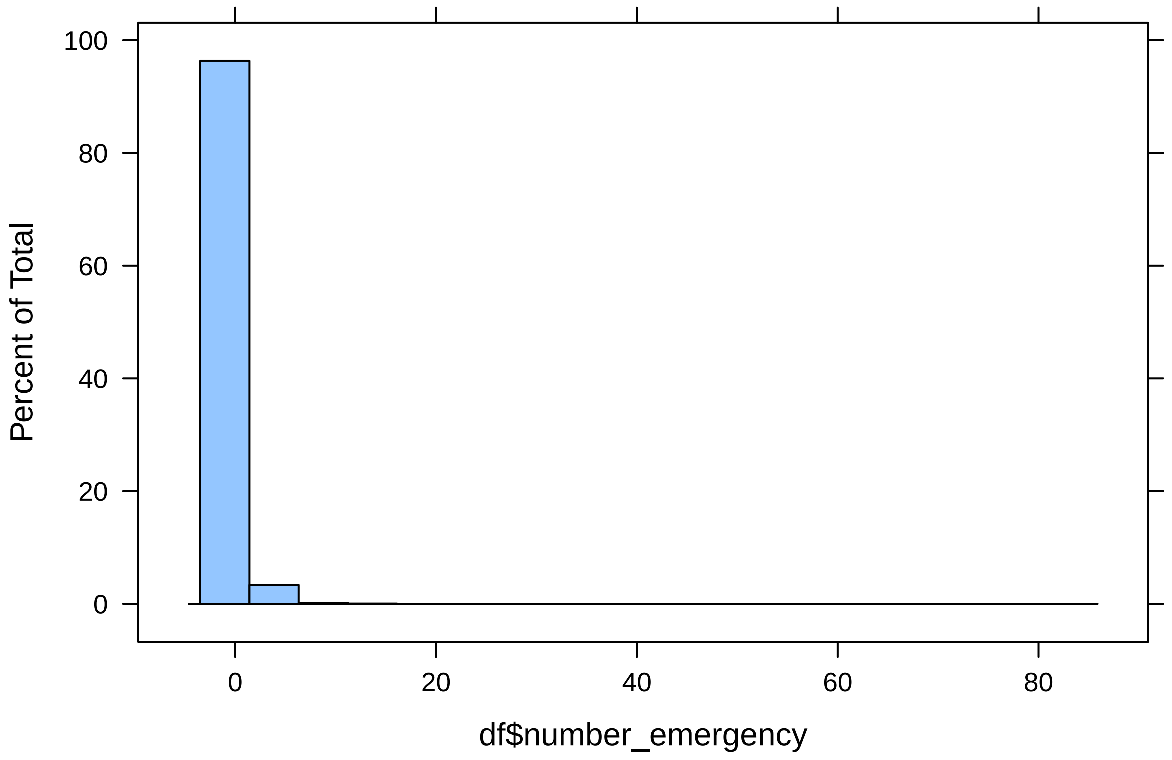
```
histogram(df$number_inpatient)
```



```
histogram(df$number_outpatient)
```



```
histogram(df$number_emergency)
```

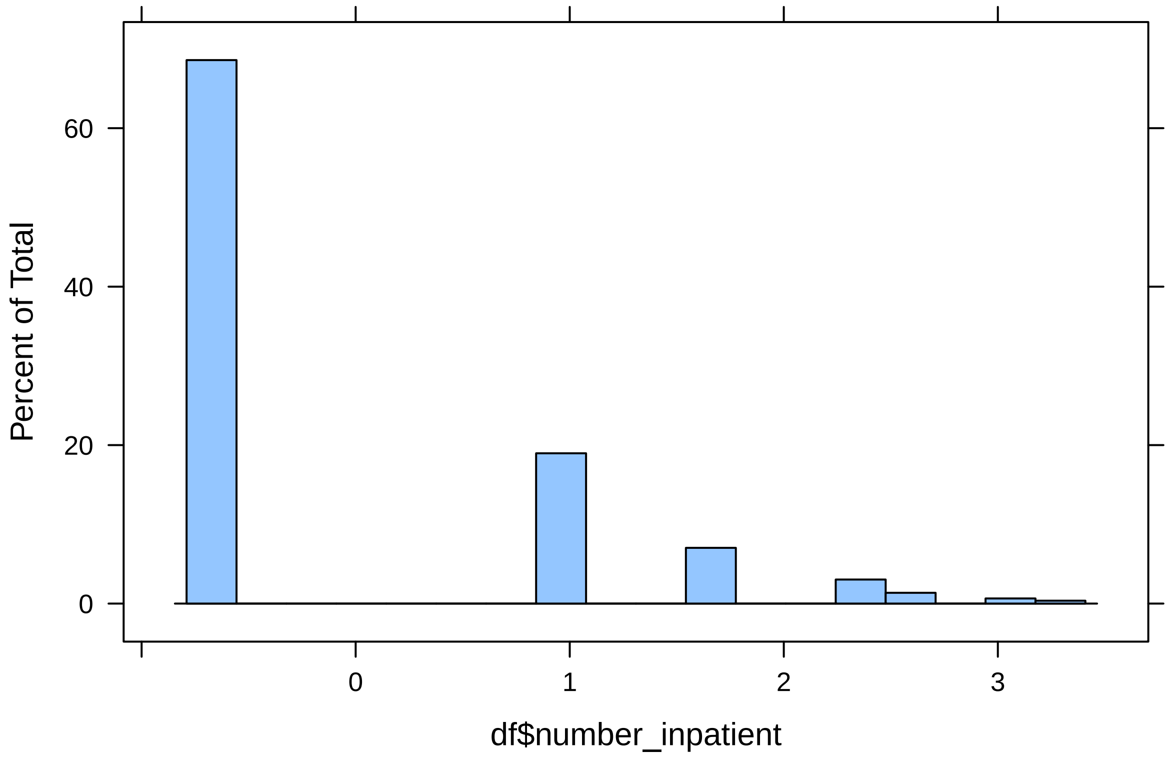


These features is needed to be normalized.

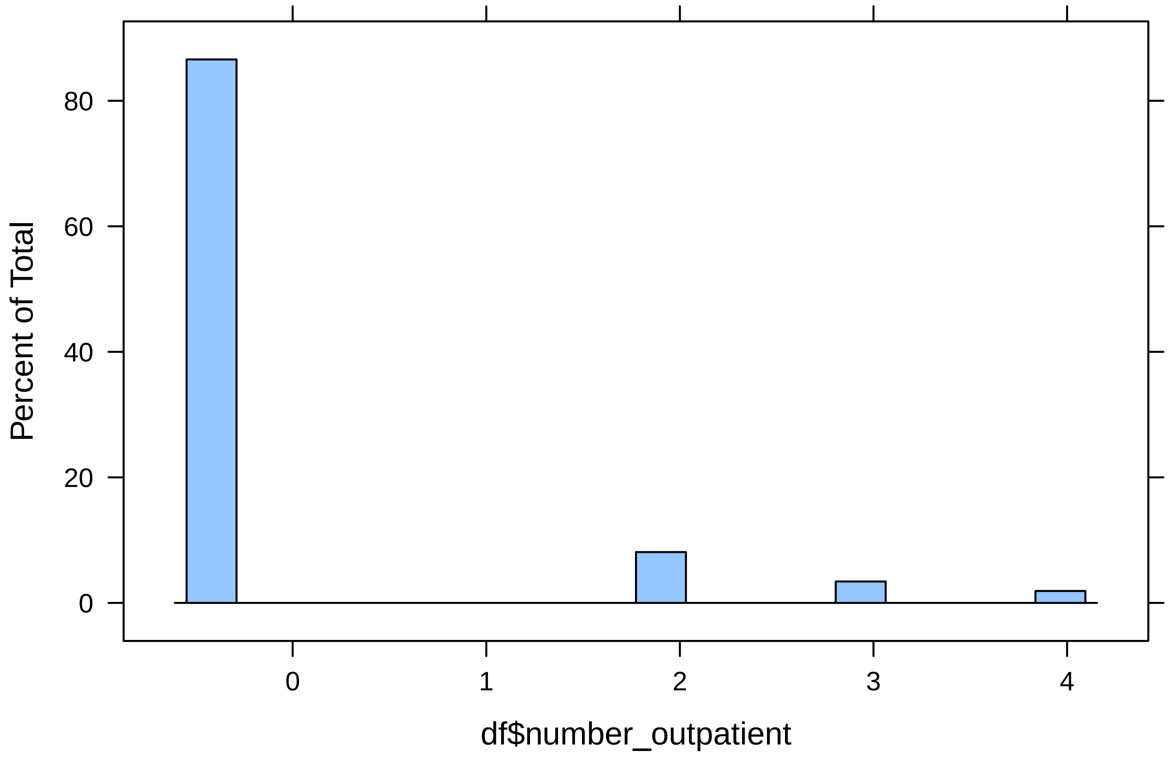
```
df$number_inpatient <- log1p(df$number_inpatient)
df$number_outpatient <- log1p(df$number_outpatient)
df$number_emergency <- log1p(df$number_emergency)
```

detecting outlier and removing it from the dataset.

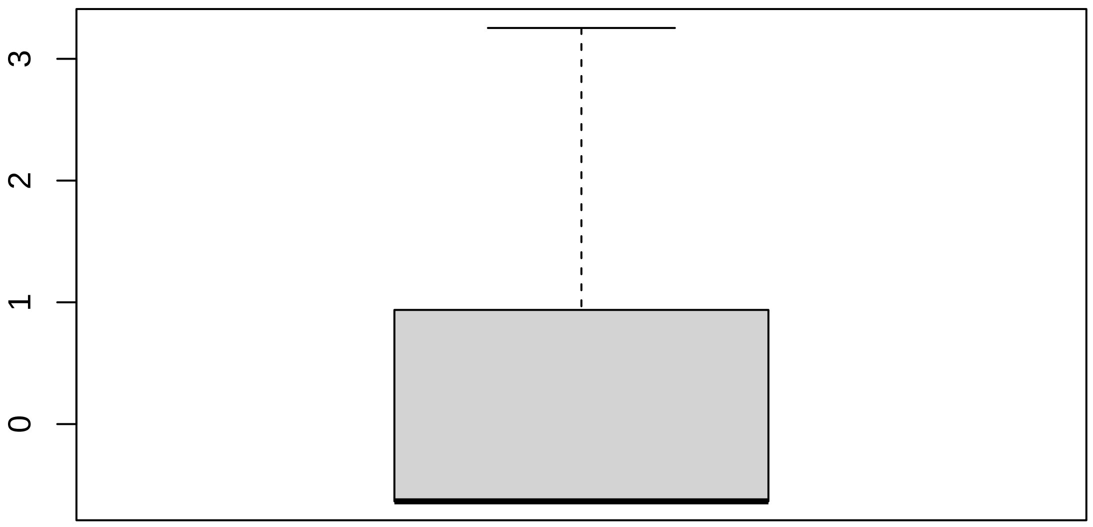
```
non_outliers = function(x, zs) {
  temp <- (x - mean(x))/sd(x)
  return(temp < zs)
}

df <- df[non_outliers(df$number_inpatient, 3),]
df <- df[non_outliers(df$number_outpatient, 3),]
df <- df[non_outliers(df$number_emergency, 3),]
df <- subset(df, select = -c(number_emergency))

#Normalise skewed features and removing outliers using z-score

cols <- dplyr::select_if(df, is.numeric)
temp <- scale(dplyr::select_if(df, is.numeric))
for(col in colnames(cols)){
  df[,col] <- temp[,col]
}
histogram(df$number_inpatient)
```



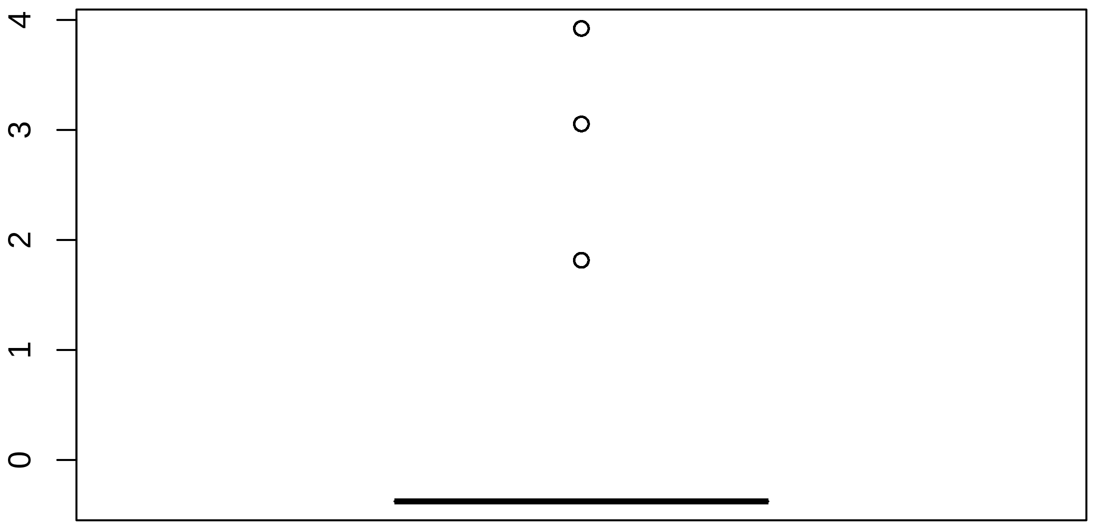
```
histogram(df$number_outpatient)
```



```
boxplot(df$number_inpatient)
```



```
boxplot(df$number_outpatient)
```



```
#make an "unknown" race category.
df$race <- ifelse(is.na(df$race), "unknown", df$race)
df$race <- as.factor(df$race)
#since there is such a large range of diagnosis' and it's nominal, will also add an "unknown" column fo
df$diag_1 <- ifelse(is.na(df$diag_1), "unknown", df$diag_1)
df$diag_2 <- ifelse(is.na(df$diag_2), "unknown", df$diag_2)
df$diag_3 <- ifelse(is.na(df$diag_3), "unknown", df$diag_3)
df <- subset(df, select = -c(medical_specialty, payer_code))
```

Handle missing values post-EDA

Split into training and testing Since we have a large dataset, we will allocate 80% to training and 20% to testing/validation.

```
trainingRows <- createDataPartition(df$readmitted, p = .8, list=FALSE)
trainingRowsAllnew <- df[trainingRows,]
testingRows<- df [-trainingRows,]
testingRows_X <- subset(testingRows, select = -readmitted)
testingRows_y <- subset(testingRows, select = readmitted)

#35,36
trainingRowsAllnew$readmitted <- factor(trainingRowsAllnew$readmitted, levels = c(0,1))


#sub-setting predictors and outcome (X and y)
X <- subset(df,select = -readmitted)
y <- subset(df,select = readmitted)

train_x <- X[trainingRows, ]
train_y <- y[trainingRows, ]
test_x <- X[-trainingRows, ]
test_y <- y[-trainingRows,]
```

Modeling

#Random Forest

```
set.seed(7)
library(randomForest)
library(caret)


indx <- createFolds(train_y, returnTrain = TRUE)
ctrl <- trainControl(method = "cv", index = indx)

mGrid <- data.frame(mtry = 2)

rfTune <- train(x=train_x, y=train_y,
                method = "rf",
                tuneGrid = mGrid,
                ntree = 20,
                importance = TRUE,
                trControl = ctrl)
predicted_labels <- predict(rfTune, test_x)
confusion_matrix_rf <- confusionMatrix(predicted_labels,test_y)
confusion_matrix_rf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 17048  2016
##          1     5     6
##
```

```
##                Accuracy : 0.894
##                  95% CI : (0.8896, 0.8984)
##     No Information Rate : 0.894
##     P-Value [Acc > NIR] : 0.4965
##
##                   Kappa : 0.0048
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.999707
##             Specificity : 0.002967
##          Pos Pred Value : 0.894251
##          Neg Pred Value : 0.545455
##              Prevalence : 0.893997
##          Detection Rate : 0.893735
##    Detection Prevalence : 0.999423
##       Balanced Accuracy : 0.501337
##
##        'Positive' Class : 0
##
```

Naive Bayes

```r
# naive bayes is not compatible with the imbalance data so balancing the data, undersampling is needed
library(dplyr)
valid_levels <- make.names(levels(train_y))
levels(train_y) <- valid_levels

# Create a data frame with train_x and train_y
train_data <- data.frame(train_x, train_y)

# Count the number of samples in each class
class_counts <- table(train_data$train_y)

# Find the class with the majority samples
majority_class <- names(class_counts)[which.max(class_counts)]

# Determine the number of samples in the minority class
minority_count <- min(class_counts)

# Perform undersampling on the majority class
undersampled_data <- train_data %>%
  group_by(train_y) %>%
  slice_sample(n = minority_count, replace = FALSE)


# Retrieve the undersampled features (train_x) and labels (train_y)
undersampled_train_x <- undersampled_data[, -ncol(undersampled_data)]
undersampled_train_y <- undersampled_data$train_y

undersampled_train_x$diag_1 <- as.factor(undersampled_train_x$diag_1)
```

```
undersampled_train_x$diag_2 <- as.factor(undersampled_train_x$diag_2)
undersampled_train_x$diag_3 <- as.factor(undersampled_train_x$diag_3)
valid_levels <- make.names(levels(test_y))
levels(test_y) <- valid_levels
```

```
model = train(undersampled_train_x,undersampled_train_y,'nb',trControl=trainControl(method='cv',number=
predicted_labels <- predict(model, test_x)
confusion_matrix_nb <- confusionMatrix(predicted_labels, test_y)
confusion_matrix_nb
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    X0    X1
##         X0 10870   811
##         X1  6183  1211
##
##                Accuracy : 0.6333
##                  95% CI : (0.6265, 0.6402)
##     No Information Rate : 0.894
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1089
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.6374
##             Specificity : 0.5989
##          Pos Pred Value : 0.9306
##          Neg Pred Value : 0.1638
##              Prevalence : 0.8940
##          Detection Rate : 0.5699
##    Detection Prevalence : 0.6124
##       Balanced Accuracy : 0.6182
##
##        'Positive' Class : X0
##
```

```
# for buiding linear model, all numeric data needed to be scaled.
numeric_cols <- sapply(df, is.numeric)

# Scale numeric columns in the dataframe
scaled_df <- df
scaled_df[, numeric_cols] <- scale(scaled_df[, numeric_cols])

trainingRows <- createDataPartition(df$readmitted, p = .8, list=FALSE)
x_train <- scaled_df[trainingRows,]
testingRows<- scaled_df [-trainingRows,]
x_test <- subset(testingRows, select = -readmitted)
y_test <- subset(testingRows, select = readmitted)

y_test$readmitted <- as.factor(y_test$readmitted )
```

```r
# Convert all columns in the dataframe to factor datatype
y_test1 <- as.data.frame(lapply(y_test, as.factor))
```

```r
set.seed(7)


lda_model <- train (readmitted ~.,data=x_train, method="lda")

pred <- predict(lda_model,x_test)
confusion_matirx_lda <- confusionMatrix(pred,y_test$readmitted)
confusion_matirx_lda
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction     0     1
##          0 16980  1978
##          1    73    44
##
##                Accuracy : 0.8925
##                  95% CI : (0.888, 0.8968)
##     No Information Rate : 0.894
##     P-Value [Acc > NIR] : 0.7566
##
##                   Kappa : 0.0299
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.99572
##             Specificity : 0.02176
##          Pos Pred Value : 0.89566
##          Neg Pred Value : 0.37607
##              Prevalence : 0.89400
##          Detection Rate : 0.89017
##    Detection Prevalence : 0.99387
##       Balanced Accuracy : 0.50874
##
##        'Positive' Class : 0
##
```

logistic regression

```r
logistic_regression <- glm(readmitted ~.,data=x_train, family = binomial)
# Make predictions using logistic regression model and threshold
threshold <- 0.3  # Set the threshold value
pred <- ifelse(predict(logistic_regression, newdata = x_test, type = "response") >= threshold, 1, 0)
pred <- as.factor(pred)
confusion_matirx_lr <- confusionMatrix(pred,y_test$readmitted)
confusion_matirx_lr
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction     0     1
##          0 16890  1943
##          1   163    79
##
##                Accuracy : 0.8896
##                  95% CI : (0.8851, 0.894)
##     No Information Rate : 0.894
##     P-Value [Acc > NIR] : 0.9761
##
##                   Kappa : 0.0482
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.99044
##             Specificity : 0.03907
##          Pos Pred Value : 0.89683
##          Neg Pred Value : 0.32645
##              Prevalence : 0.89400
##          Detection Rate : 0.88545
##    Detection Prevalence : 0.98731
##       Balanced Accuracy : 0.51476
##
##        'Positive' Class : 0
##
```

Penalized Logistic Regression

```
library(glmnet)

scaled_cols_train <- x_train[,sapply(x_train, is.numeric)]
scaled_cols_test <- x_test [,sapply(x_test, is.numeric)]
glmnet_model <- cv.glmnet(as.matrix(scaled_cols_train), x_train$readmitted, family="binomial", type.meas
pred <- predict (glmnet_model,newx= as.matrix(scaled_cols_test, type= "response"))
prediction <- ifelse(pred>0.5, 1, 0)
pred <- as.factor(prediction)

confusion_matirx_glmnet <- confusionMatrix(pred,y_test$readmitted)
confusion_matirx_glmnet
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 17053  2022
##          1     0     0
##
##                Accuracy : 0.894
##                  95% CI : (0.8895, 0.8983)
##     No Information Rate : 0.894
##     P-Value [Acc > NIR] : 0.5059
```

```
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 1.000
##             Specificity : 0.000
##          Pos Pred Value : 0.894
##          Neg Pred Value :   NaN
##              Prevalence : 0.894
##          Detection Rate : 0.894
##    Detection Prevalence : 1.000
##       Balanced Accuracy : 0.500
##
##        'Positive' Class : 0
##
```

```
# Assuming you have the following variables:
# confusion_Matrix, confusion_matirx_lda, confusion_matrix_nb (from previous code)

# Calculate F1 score for each model
f1_score <- c(
  2 * confusion_matrix_rf$byClass["Pos Pred Value"] * confusion_matrix_rf$byClass["Sensitivity"] /
    (confusion_matrix_rf$byClass["Pos Pred Value"] + confusion_matrix_rf$byClass["Sensitivity"]),
  2 * confusion_matirx_lda$byClass["Pos Pred Value"] * confusion_matirx_lda$byClass["Sensitivity"] /
    (confusion_matirx_lda$byClass["Pos Pred Value"] + confusion_matirx_lda$byClass["Sensitivity"]),
  2 * confusion_matirx_nb$byClass["Pos Pred Value"] * confusion_matirx_nb$byClass["Sensitivity"] /
    (confusion_matirx_nb$byClass["Pos Pred Value"] + confusion_matirx_nb$byClass["Sensitivity"]),
  2 * confusion_matirx_lr$byClass["Pos Pred Value"] * confusion_matirx_lr$byClass["Sensitivity"] /
    (confusion_matirx_lr$byClass["Pos Pred Value"] + confusion_matirx_lr$byClass["Sensitivity"]),
  2 * confusion_matirx_glmnet$byClass["Pos Pred Value"] * confusion_matirx_glmnet$byClass["Sensitivity"]
    (confusion_matirx_glmnet$byClass["Pos Pred Value"] + confusion_matirx_glmnet$byClass["Sensitivity"]
)

# Calculate AUC area for each model
```

```
#auc_area
confusion_matirx_glmnet$auc_area <- (confusion_matirx_glmnet$table[1, 1] + confusion_matirx_glmnet$tabl
    (sum(confusion_matirx_glmnet$table))

confusion_matirx_lr$auc_area <- (confusion_matirx_lr$table[1, 1] + confusion_matirx_lr$table[2, 2]) /(s

confusion_matirx_lda$auc_area <- (confusion_matirx_lda$table[1, 1] + confusion_matirx_lda$table[2, 2])
    (sum(confusion_matirx_lda$table))

confusion_matrix_rf$auc_area <- (confusion_matrix_rf$table[1, 1] + confusion_matrix_rf$table[2, 2]) /
    (sum(confusion_matrix_rf$table))

confusion_matrix_nb$auc_area <- (confusion_matrix_nb$table["X0", "X0"] + confusion_matrix_nb$table["X1"
```

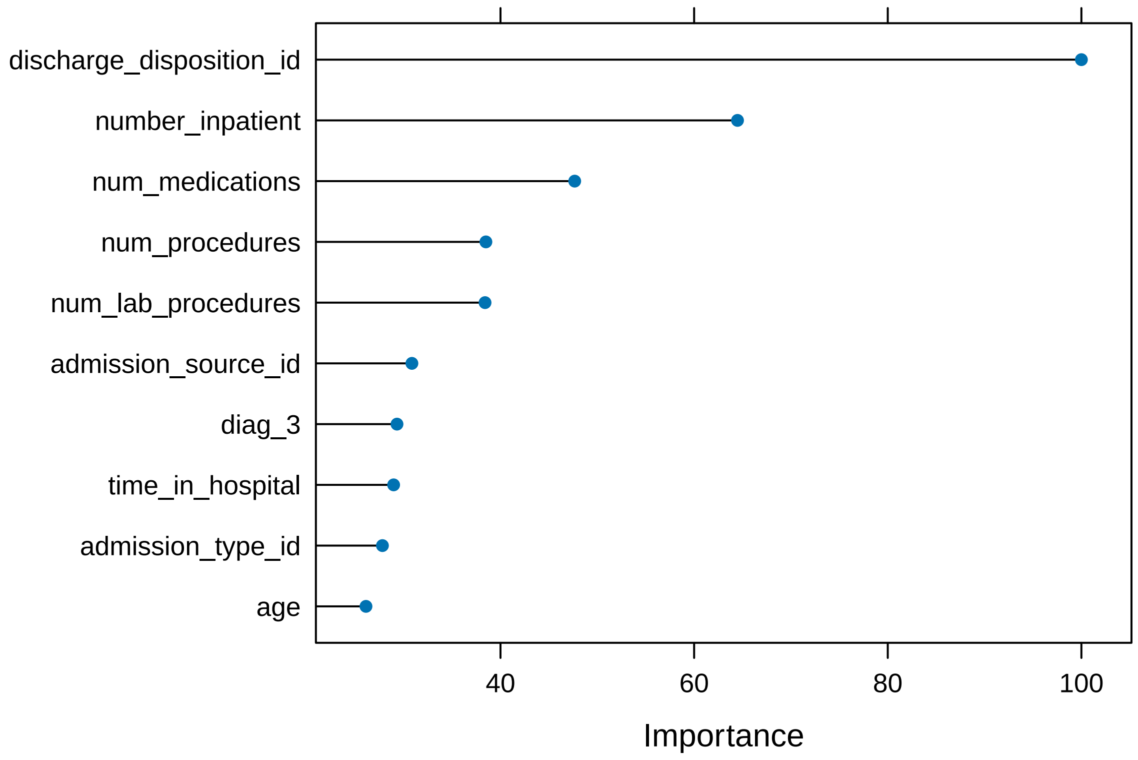
```
## Calculate f1_score for each model
```

```
confusion_matrix_rf$f1_score <- 2 * confusion_matrix_rf$byClass["Pos Pred Value"] * confusion_matrix_rf$
    (confusion_matrix_rf$byClass["Pos Pred Value"] + confusion_matrix_rf$byClass["Sensitivity"])

confusion_matirx_lda$f1_score <- 2 * confusion_matirx_lda$byClass["Pos Pred Value"] * confusion_matirx_l
    (confusion_matirx_lda$byClass["Pos Pred Value"] + confusion_matirx_lda$byClass["Sensitivity"])

confusion_matrix_nb$f1_score <- 2 * confusion_matrix_nb$byClass["Pos Pred Value"] * confusion_matrix_nb$
    (confusion_matrix_nb$byClass["Pos Pred Value"] + confusion_matrix_nb$byClass["Sensitivity"])

confusion_matirx_lr$f1_score <- 2 * confusion_matirx_lr$byClass["Pos Pred Value"] * confusion_matirx_lr$
    (confusion_matirx_lr$byClass["Pos Pred Value"] + confusion_matirx_lr$byClass["Sensitivity"])

confusion_matirx_glmnet$f1_score <- 2 * confusion_matirx_glmnet$byClass["Pos Pred Value"] * confusion_ma
    (confusion_matirx_glmnet$byClass["Pos Pred Value"] + confusion_matirx_glmnet$byClass["Sensitivity"]
```

```
# Calculate accuracy for each model
accuracy <- c(confusion_matirx_glmnet$overall["Accuracy"], confusion_matirx_lr$overall["Accuracy"], con
confusion_matrix_nb$overall["Accuracy"],
confusion_matrix_rf$overall["Accuracy"])


# Calculate AUC area for each model
auc_area <- c(confusion_matirx_glmnet$auc_area,confusion_matirx_lr$auc_area ,confusion_matirx_lda$auc_a

# Calculate f1_score for each model

f1_score <- c(confusion_matirx_glmnet$f1_score,confusion_matirx_lr$f1_score,confusion_matirx_lda$f1_sco



# Create a data frame with the evaluation metrics
model_evaluation <- data.frame(
  Model = c("Penalized Logistic Regression", "Logistic Regression", "Linear Discriminant Analysis", "Na
  Accuracy = accuracy,
  F1_Score = f1_score,
  AUC_Area = auc_area
)

model_evaluation
```

```
##                           Model  Accuracy  F1_Score  AUC_Area
## 1 Penalized Logistic Regression 0.8939974 0.9440323 0.8939974
## 2           Logistic Regression 0.8895937 0.9413142 0.8895937
## 3  Linear Discriminant Analysis 0.8924771 0.9430452 0.8924771
## 4                   Naive Bayes 0.6333421 0.7565950 0.6333421
## 5                 random forest 0.8940498 0.9440430 0.8940498
```

```
lrImp_random_forest <- varImp(rfTune, scale = TRUE)
plot(lrImp_random_forest,top = 10)
```



Top predictors of the model are: Discha Discharge: disposition Nominal Integer identifier corresponding to 29 distinct values, for example, discharged to home, expired, and not available. number_impatient: Number of inpatient visits of the patient in the year preceding the encounter. admission_source_id: Admission sources are physician referral, emergency room, and transfer from a hospital.