Diabetes Readmission Prediction

Vivian Do, Bethany Wang

2023-06-24

1. Initial Data Preparation

1.1 Import and check the Data

```
#Read in data, replace "?" with NA values
df <- read.csv(file = "diabetic_data.csv", na.strings=c("?"))
dim(df)
```

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

```
str(df)
```

```
## 'data.frame':    101766 obs. of  50 variables:
##  $ encounter_id            : int  2278392 149190 64410 500364 16680 35754 55842 63768 1252
2 15738 ...
##  $ patient_nbr             : int  8222157 55629189 86047875 82442376 42519267 82637451 842
59809 114882984 48330783 63555939 ...
##  $ race                    : chr  "Caucasian" "Caucasian" "AfricanAmerican" "Caucasian"
...
##  $ gender                  : chr  "Female" "Female" "Female" "Male" ...
##  $ age                     : chr  "[0-10)" "[10-20)" "[20-30)" "[30-40)" ...
##  $ weight                  : chr  NA NA NA NA ...
##  $ admission_type_id       : int  6 1 1 1 1 2 3 1 2 3 ...
##  $ discharge_disposition_id: int  25 1 1 1 1 1 1 1 1 3 ...
##  $ admission_source_id     : int  1 7 7 7 7 2 2 7 4 4 ...
##  $ time_in_hospital        : int  1 3 2 2 1 3 4 5 13 12 ...
##  $ payer_code              : chr  NA NA NA NA ...
##  $ medical_specialty       : chr  "Pediatrics-Endocrinology" NA NA NA ...
##  $ num_lab_procedures      : int  41 59 11 44 51 31 70 73 68 33 ...
##  $ num_procedures          : int  0 0 5 1 0 6 1 0 2 3 ...
##  $ num_medications         : int  1 18 13 16 8 16 21 12 28 18 ...
##  $ number_outpatient       : int  0 0 2 0 0 0 0 0 0 0 ...
##  $ number_emergency        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ number_inpatient        : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ diag_1                  : chr  "250.83" "276" "648" "8" ...
##  $ diag_2                  : chr  NA "250.01" "250" "250.43" ...
##  $ diag_3                  : chr  NA "255" "V27" "403" ...
##  $ number_diagnoses        : int  1 9 6 7 5 9 7 8 8 8 ...
##  $ max_glu_serum           : chr  "None" "None" "None" "None" ...
##  $ A1Cresult               : chr  "None" "None" "None" "None" ...
##  $ metformin               : chr  "No" "No" "No" "No" ...
##  $ repaglinide             : chr  "No" "No" "No" "No" ...
##  $ nateglinide             : chr  "No" "No" "No" "No" ...
##  $ chlorpropamide          : chr  "No" "No" "No" "No" ...
##  $ glimepiride             : chr  "No" "No" "No" "No" ...
##  $ acetohexamide           : chr  "No" "No" "No" "No" ...
##  $ glipizide               : chr  "No" "No" "Steady" "No" ...
##  $ glyburide               : chr  "No" "No" "No" "No" ...
##  $ tolbutamide             : chr  "No" "No" "No" "No" ...
##  $ pioglitazone            : chr  "No" "No" "No" "No" ...
##  $ rosiglitazone           : chr  "No" "No" "No" "No" ...
##  $ acarbose                : chr  "No" "No" "No" "No" ...
##  $ miglitol                : chr  "No" "No" "No" "No" ...
##  $ troglitazone            : chr  "No" "No" "No" "No" ...
##  $ tolazamide              : chr  "No" "No" "No" "No" ...
##  $ examide                 : chr  "No" "No" "No" "No" ...
##  $ citoglipton             : chr  "No" "No" "No" "No" ...
##  $ insulin                 : chr  "No" "Up" "No" "Up" ...
##  $ glyburide.metformin     : chr  "No" "No" "No" "No" ...
##  $ glipizide.metformin     : chr  "No" "No" "No" "No" ...
##  $ glimepiride.pioglitazone: chr  "No" "No" "No" "No" ...
##  $ metformin.rosiglitazone : chr  "No" "No" "No" "No" ...
##  $ metformin.pioglitazone  : chr  "No" "No" "No" "No" ...
##  $ change                  : chr  "No" "Ch" "No" "Ch" ...
##  $ diabetesMed             : chr  "No" "Yes" "Yes" "Yes" ...
##  $ readmitted              : chr  "NO" ">30" "NO" "NO" ...
```

- There are 101766 data records with 50 features

1.2 Filter out Irrelevant Observations

- All patient encounters associated with a discharge to hospice or death will be removed, as the chance of readmission is low-impossible.

```
dischargedRemoved <- c(11,13,14,19,20,21)
df <- subset(df, !(discharge_disposition_id %in% dischargedRemoved))
```

- encounter_id and patient_nbr are identifiers, not useful features. Payer_code is irrelevant feature too. They will be removed.

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

```
## [1] 99343    47
```

1.3 Filter out degenerated features

```
library(caret)

# Use nearZeroVar function to filter out low variance features
degenerateCols <- nearZeroVar(df)
length(degenerateCols)
```

```
## [1] 18
```

```
degenerateColNames <- colnames(df[degenerateCols])
degenerateColNum <- length(degenerateColNames)

cat("These are ", degenerateColNum, " degenerated predictors:\n", degenerateColNames, "\n\n")
```

```
## These are  18  degenerated predictors:
##  max_glu_serum repaglinide nateglinide chlorpropamide glimepiride acetohexamide tolbutamid
e acarbose miglitol troglitazone tolazamide examide citoglipton glyburide.metformin glipizid
e.metformin glimepiride.pioglitazone metformin.rosiglitazone metformin.pioglitazone
```

```
df<- df[, -degenerateCols]
dim(df)
```

```
## [1] 99343    29
```

```
colnames(df)
```

```
##  [1] "race"                    "gender"
##  [3] "age"                     "weight"
##  [5] "admission_type_id"       "discharge_disposition_id"
##  [7] "admission_source_id"     "time_in_hospital"
##  [9] "medical_specialty"       "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"        "A1Cresult"
## [21] "metformin"               "glipizide"
## [23] "glyburide"               "pioglitazone"
## [25] "rosiglitazone"           "insulin"
## [27] "change"                  "diabetesMed"
## [29] "readmitted"
```

- 18 degenerated predictors that show near-zero variance have been removed. 29 columns are left in the dataset.

1.4 Filter out categorical levels indicating nulls

'discharge_disposition_id' (discharge ID), 'admission_type_id' (admission type ID), and 'admission_source_id' (admission source ID) are categorical variables with levels identified by their ID number. For example, 'admission_type_id' 1 refers to 'Emergency'.

For admission source and type, the following ID levels associated with null values will be replaced as NA:

- Admission source ID 9 (Not Available), 17 (NULL), and 20 (Not Mapped)
- Admission type ID 6 (NULL) and 8 (Not Mapped)

```
admissionsourceRemoved <- c(9,17,20)
df$admission_source_id[df$admission_source_id %in% admissionsourceRemoved] <- NA

admissiontypeRemoved <- c(6,8)
df$admission_type_id[df$admission_type_id %in% admissiontypeRemoved] <- NA
table(df$admission_source_id)
```

```
##
##     1     2     3     4     5     6     7     8    10    11    13    14    22
## 29168  1081   185  3118   806  2239 55850    15     8     2     1     2    12
##    25
##     2
```

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

```
## [1] 6854
```

1.5 Handle missing values

- Check missing values

```
# Check null values
#Show null counts
null_counts <- sort(colSums(is.na(df)), decreasing=TRUE)
head(null_counts,10)
```

```
##              weight   medical_specialty admission_source_id   admission_type_id
##               96218               48616                6854                5527
##                race              diag_3              diag_2              diag_1
##                2234                1419                 356                  20
##              gender                 age
##                   0                   0
```

```
#Show proportion of null counts
nullProp <- sort(round(colMeans(is.na(df)),3), decreasing=TRUE)
head(nullProp,10)
```

```
##                   weight        medical_specialty      admission_source_id
##                    0.969                    0.489                    0.069
##        admission_type_id                     race                   diag_3
##                    0.056                    0.022                    0.014
##                   diag_2                   gender                      age
##                    0.004                    0.000                    0.000
## discharge_disposition_id
##                    0.000
```

- The following predictors contain a significant number of null values: 'weight' (96.9%), medical_specialty (48.9%). These two predictors will be removed.
- The following predictors contain less than 2% null values: 'race' (2.2%), 'diag_3' (1.4%), 'diag_2' (0.4%), and 'diag_1' (0.02%). Because the null percentages are very low, we will simply remove the rows with null values.

```
# remove weight and medical_specialty
df<- subset(df, select = -c(weight, medical_specialty))

# Remove other null values
df <- na.omit(df)

null_counts <- sort(colSums(is.na(df)), decreasing=TRUE)
head(null_counts,10)
```

```
##                     race                   gender                      age
##                        0                        0                        0
##        admission_type_id discharge_disposition_id      admission_source_id
##                        0                        0                        0
##         time_in_hospital       num_lab_procedures           num_procedures
##                        0                        0                        0
##          num_medications
##                        0
```

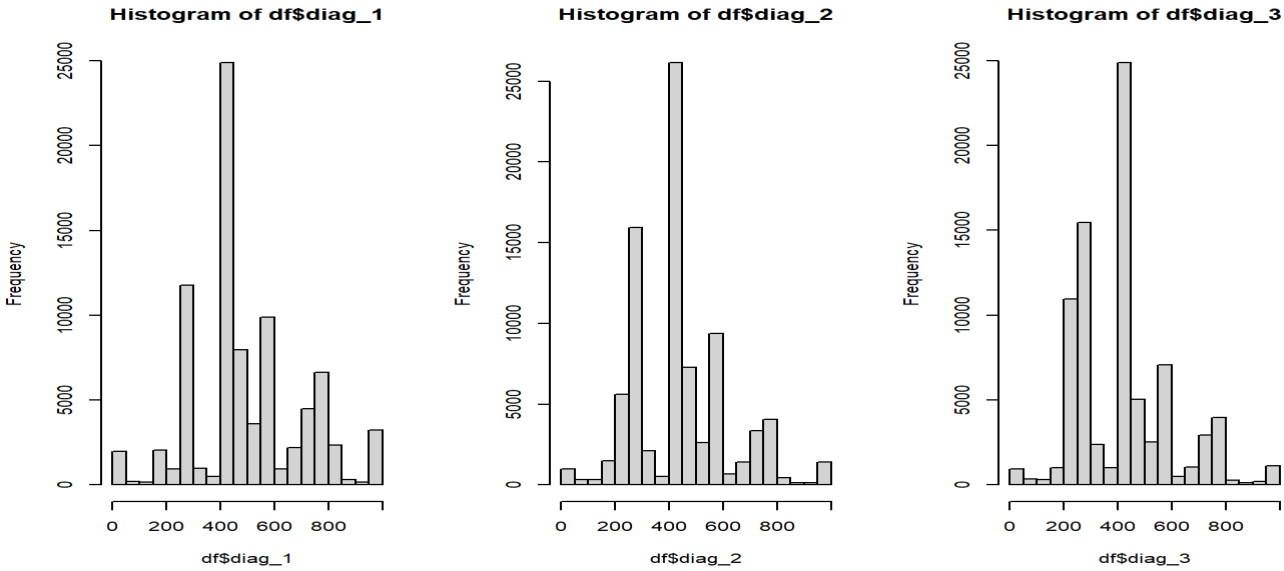
```
dim(df)
```

```
## [1] 86566    27
```

1.6 Feature Creation

- Explore primary diagnoses ('diag_1') and secondary diagnoses ('diag_2', 'diag_3')

```
# Convert to numeric
df$diag_1 <- as.numeric(df$diag_1)
df$diag_2 <- as.numeric(df$diag_2)
df$diag_3 <- as.numeric(df$diag_3)

# Show distributions of diagnoses
par(mfrow=c(1,3))
hist(df$diag_1)
hist(df$diag_2)
hist(df$diag_3)
```




```
#summary(df$diag_1)
```

- Extract primary/secondary diagnoses

```r
library(dplyr)

df <- df %>%
  mutate(primary_diagnosis = case_when(
    substr(diag_1, 1, 3) == "250" ~ "Diabetes",
    #between(as.numeric(diag_1), 001, 139) ~ "Infectious and Parasitic Diseases",
    between(as.numeric(diag_1), 140, 239) ~ "Neoplasms",
    between(as.numeric(diag_1), 320, 459) ~ "Circulatory",
    between(as.numeric(diag_1), 460, 519) ~ "Respiratory",
    between(as.numeric(diag_1), 520, 579) ~ "Digestive",
    between(as.numeric(diag_1), 580, 629) ~ "Genitourinary System",
    #between(as.numeric(diag_1), 680, 709) ~ "Skin/Subcutaneous Tissue",
    between(as.numeric(diag_1), 710, 739) ~ "Musculoskeletal",
    between(as.numeric(diag_1), 760, 779) ~ "Perinatal",
    between(as.numeric(diag_1), 800, 999) ~ "Injury and Poisoning",
    TRUE ~ "Other"
  ))

# Secondary Diagnosis
df <- df %>%
  mutate(secondary_diagnosis = case_when(
    substr(diag_2, 1, 3) == "250" ~ "Diabetes",
    #between(as.numeric(diag_2), 001, 139) ~ "Infectious and Parasitic Diseases",
    between(as.numeric(diag_2), 140, 239) ~ "Neoplasms",
    between(as.numeric(diag_2), 320, 459) ~ "Circulatory",
    between(as.numeric(diag_2), 460, 519) ~ "Respiratory",
    between(as.numeric(diag_2), 520, 579) ~ "Digestive",
    between(as.numeric(diag_2), 580, 629) ~ "Genitourinary System",
    #between(as.numeric(diag_2), 680, 709) ~ "Skin/Subcutaneous Tissue",
    between(as.numeric(diag_2), 710, 739) ~ "Musculoskeletal",
    between(as.numeric(diag_2), 760, 779) ~ "Perinatal",
    between(as.numeric(diag_2), 800, 999) ~ "Injury and Poisoning",
    TRUE ~ "Other"
  ))

# Secondary Diagnosis
df <- df %>%
  mutate(secondary_diagnosis2 = case_when(
    substr(diag_3, 1, 3) == "250" ~ "Diabetes",
    #between(as.numeric(diag_3), 001, 139) ~ "Infectious and Parasitic Diseases",
    between(as.numeric(diag_3), 140, 239) ~ "Neoplasms",
    between(as.numeric(diag_3), 320, 459) ~ "Circulatory",
    between(as.numeric(diag_3), 460, 519) ~ "Respiratory",
    between(as.numeric(diag_3), 520, 579) ~ "Digestive",
    between(as.numeric(diag_3), 580, 629) ~ "Genitourinary System",
    #between(as.numeric(diag_3), 680, 709) ~ "Skin/Subcutaneous Tissue",
    between(as.numeric(diag_3), 710, 739) ~ "Musculoskeletal",
    between(as.numeric(diag_3), 760, 779) ~ "Perinatal",
    between(as.numeric(diag_3), 800, 999) ~ "Injury and Poisoning",
    TRUE ~ "Other"
  ))
df$primary_diagnosis <- as.factor(df$primary_diagnosis)
df$secondary_diagnosis <- as.factor(df$secondary_diagnosis)
df$secondary_diagnosis2 <- as.factor(df$secondary_diagnosis2)
```

- Show counts of diseases for the primary and secondary diagnosis:

```
table(df$primary_diagnosis)
```

```
##
##          Circulatory             Diabetes            Digestive
##                27207                 7185                 7873
## Genitourinary System Injury and Poisoning      Musculoskeletal
##                 4445                 6017                 4217
##            Neoplasms                Other          Respiratory
##                 2762                18242                 8618
```

```
# table(df$secondary_diagnosis)
# table(df$secondary_diagnosis2)
```

- All levels w/ less than 2500 instances are combined into 'Other'
- Most common primary/secondary diagnosis were circulatory.
- 7185 patients had diabetes as their primary diagnosis
- diag_1, diag_2, and diag_3 are turned to new variables. They are not needed anymore and will be removed.

```
df <- subset(df, select = -c(diag_1, diag_2,diag_3))
dim(df)
```

```
## [1] 86566    27
```

1.7 Convert categorical variables into factors

- All feature that are recorded as chr type are categorical variables
- admission_type_id, discharge_disposition_id, admission_source_id are recorded as int, but should be taken as categorical variables.

```
#Convert categorical variables into factors (according to Table 1 https://www.hindawi.com/jou
rnals/bmri/2014/781670/tab1/)
df[sapply(df, is.character)] <- lapply(df[sapply(df, is.character)],
                                       as.factor)

df$admission_type_id <- as.factor(df$admission_type_id)
df$discharge_disposition_id <- as.factor(df$discharge_disposition_id)
df$admission_source_id <- as.factor(df$admission_source_id)
```

- All categorical variables have been converted into factors

1.8 Classify features

- 27 variables are left for further analysis.
- Subset numerical and categorical features

```
# Subset numerical and categorical features into new data frames
library(dplyr)
df_num <- df %>%  select_if(is.integer)
df_cat <- df %>% select_if(is.factor)

# Show numerical variables
cat("These are the numerical features:\n", colnames(df_num), "\n\n")
```

```
## These are the numerical features:
##  time_in_hospital num_lab_procedures num_procedures num_medications number_outpatient numb
er_emergency number_inpatient number_diagnoses
```

```
# Show categorical variables
cat("These are the categorical features:\n", colnames(df_cat), "\n\n")
```

```
## These are the categorical features:
##  race gender age admission_type_id discharge_disposition_id admission_source_id A1Cresult
metformin glipizide glyburide pioglitazone rosiglitazone insulin change diabetesMed readmitte
d primary_diagnosis secondary_diagnosis secondary_diagnosis2
```

1.9 Explore and revalue the response variable

- We use “readmitted” as our response variable

```
# Show readmission counts
table(df$readmitted)
```

```
##
##   <30   >30    NO
##  9985 31109 45472
```

We will combine all patients who were admitted into one level. Thus, our response variable will be binary:

- YES if the patient was readmitted at any time
- NO if the patient was not readmitted

```
library(plyr)

revalue(df$readmitted, c("<30" = "YES")) -> df$readmitted
revalue(df$readmitted, c(">30" = "YES")) -> df$readmitted

# Show readmission counts using binary levels
table(df$readmitted)
```

```
##
##   YES    NO
## 41094 45472
```

```
# Show readmission ratio using binary levels
round(table(df$readmitted) / length(df$readmitted),2)
```

```
##
##  YES   NO
## 0.47 0.53
```

- The response variable now contains 47% of “YES” and 53% of “NO”.

2. Exploratory Data Analysis

2.1 Statistics of the numerical variables

```
summary(df_num)
```

```
##  time_in_hospital num_lab_procedures num_procedures  num_medications
##  Min.   : 1.000   Min.   :  1.00     Min.   :0.000   Min.   : 1.00
##  1st Qu.: 2.000   1st Qu.: 33.00     1st Qu.:0.000   1st Qu.:10.00
##  Median : 4.000   Median : 44.00     Median :1.000   Median :15.00
##  Mean   : 4.413   Mean   : 43.42     Mean   :1.348   Mean   :16.06
##  3rd Qu.: 6.000   3rd Qu.: 57.00     3rd Qu.:2.000   3rd Qu.:20.00
##  Max.   :14.000   Max.   :132.00     Max.   :6.000   Max.   :81.00
##  number_outpatient number_emergency  number_inpatient  number_diagnoses
##  Min.   : 0.0000   Min.   : 0.0000   Min.   : 0.0000   Min.   : 3.000
##  1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.: 0.0000   1st Qu.: 6.000
##  Median : 0.0000   Median : 0.0000   Median : 0.0000   Median : 9.000
##  Mean   : 0.3664   Mean   : 0.2017   Mean   : 0.6462   Mean   : 7.543
##  3rd Qu.: 0.0000   3rd Qu.: 0.0000   3rd Qu.: 1.0000   3rd Qu.: 9.000
##  Max.   :42.0000   Max.   :76.0000   Max.   :21.0000   Max.   :16.000
```

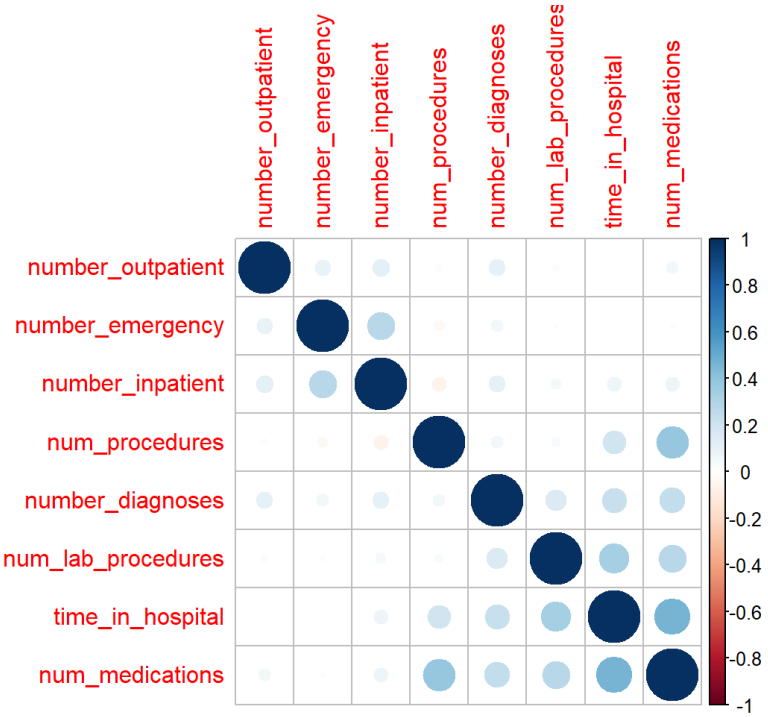
2.2 Correlations of the numerical variables

- Find Correlations of numerical features

```
library(corrplot)
correlations <- round(cor(df_num), 2)
correlations
```

```
##                    time_in_hospital num_lab_procedures num_procedures
## time_in_hospital               1.00               0.33           0.19
## num_lab_procedures             0.33               1.00           0.03
## num_procedures                 0.19               0.03           1.00
## num_medications                0.46               0.27           0.38
## number_outpatient             -0.01               0.02          -0.02
## number_emergency              -0.01               0.01          -0.04
## number_inpatient               0.07               0.04          -0.07
## number_diagnoses               0.22               0.16           0.05
##                    num_medications number_outpatient number_emergency
## time_in_hospital              0.46             -0.01            -0.01
## num_lab_procedures            0.27              0.02             0.01
## num_procedures                0.38             -0.02            -0.04
## num_medications               1.00              0.05             0.01
## number_outpatient             0.05              1.00             0.09
## number_emergency              0.01              0.09             1.00
## number_inpatient              0.07              0.11             0.27
## number_diagnoses              0.24              0.10             0.05
##                    number_inpatient number_diagnoses
## time_in_hospital               0.07             0.22
## num_lab_procedures             0.04             0.16
## num_procedures                -0.07             0.05
## num_medications                0.07             0.24
## number_outpatient              0.11             0.10
## number_emergency               0.27             0.05
## number_inpatient               1.00             0.10
## number_diagnoses               0.10             1.00
```

```
# Correlation plot
corrplot(correlations, order = "hclust")
```



Observing the correlation table and heatmap, there is modest correlations between:

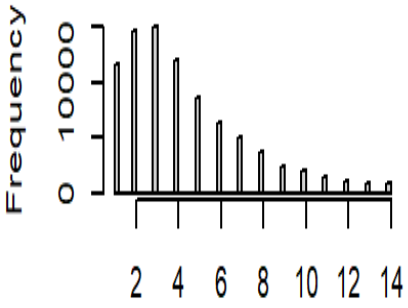
- num_lab_procedures and time_in_hospital (0.33)
- num_medications and and time_in_hospital (0.46)

- num_lab_procedures and num_medications (0.38)

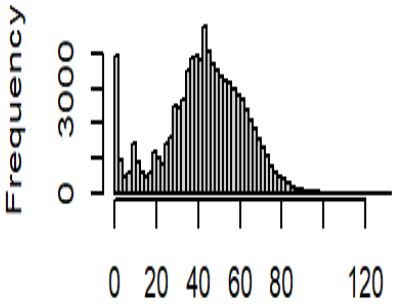
Other than that, there is no much correlations between other variables.

2.3 Distribution of the numerical variables

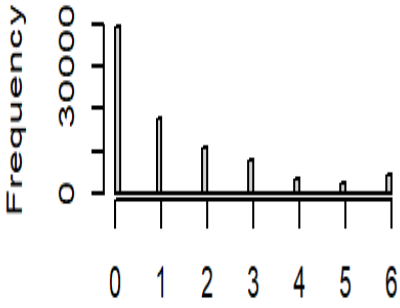
```
library(Hmisc)
hist.data.frame(df_num)
```



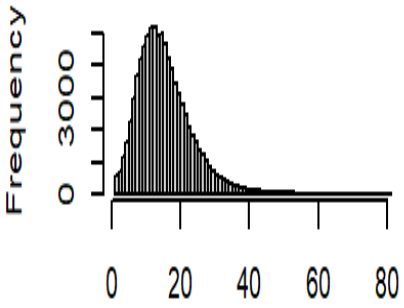
n:86566 m:0



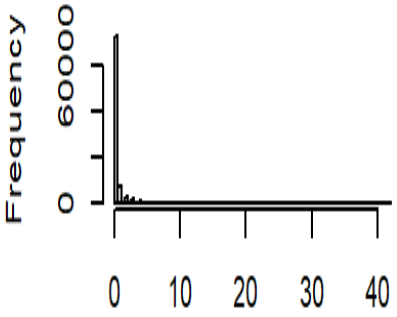
n:86566 m:0



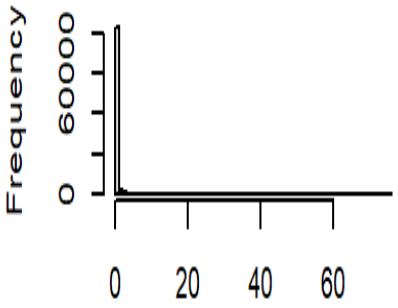
n:86566 m:0



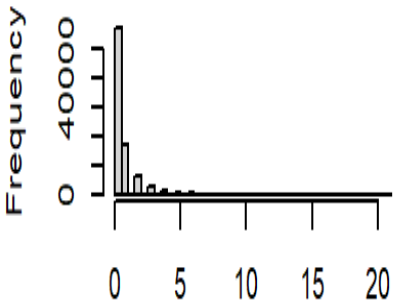
n:86566 m:0



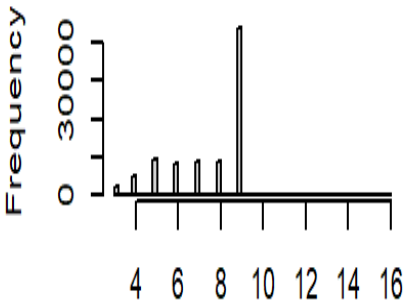
n:86566 m:0



n:86566 m:0



n:86566 m:0

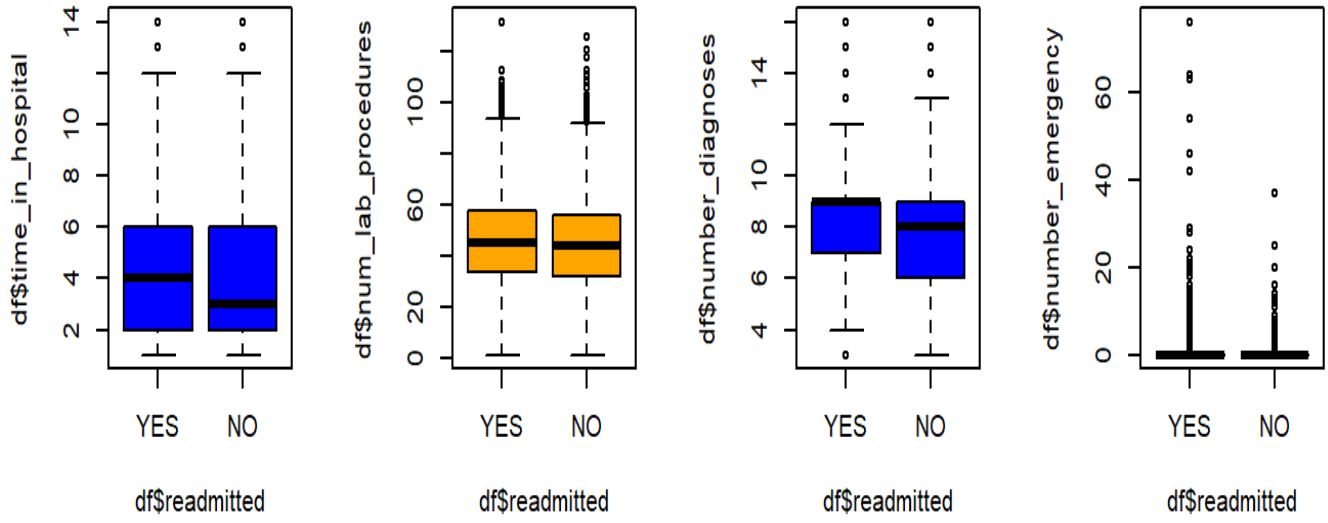


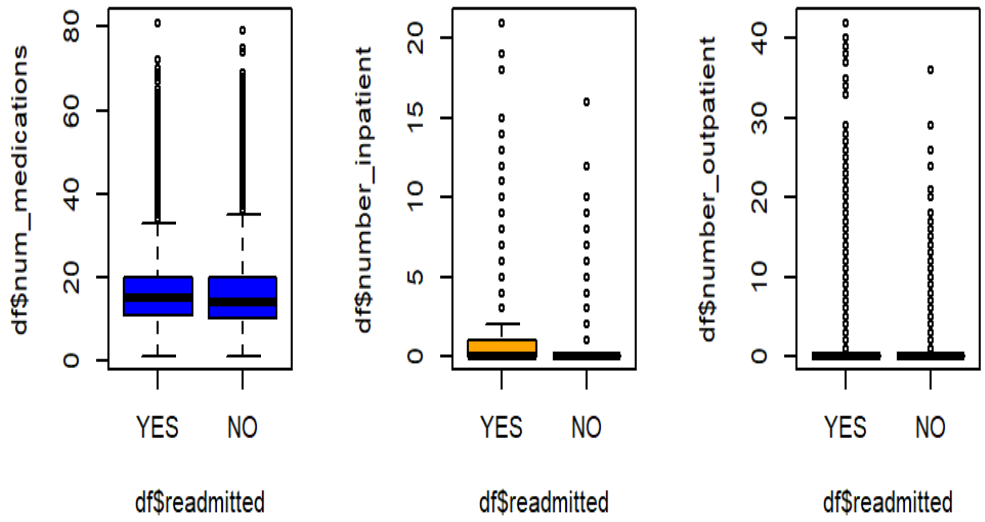
n:86566 m:0

- number_medications and number_lab_procedures have a distribution that is close to normal distribution. The distribution for other variables are all skewed.

2.4 Explore the relationship between the numerical predictors and the predicted variable

```
par(mfrow = c(2,4))

boxplot(df$time_in_hospital ~ df$readmitted, col="blue")
boxplot(df$num_lab_procedures ~ df$readmitted, col="orange")
boxplot(df$number_diagnoses ~ df$readmitted, col="blue")

boxplot(df$number_emergency ~ df$readmitted, col="orange")
boxplot(df$num_medications ~ df$readmitted, col="blue")
boxplot(df$number_inpatient ~ df$readmitted, col="orange")
boxplot(df$number_outpatient ~ df$readmitted, col="blue")
```



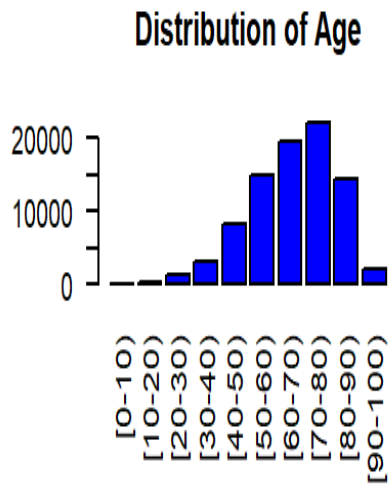


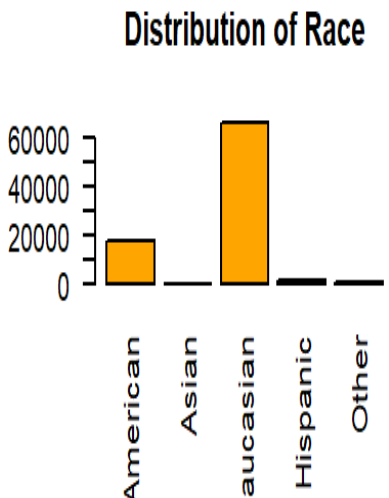
Patients who are readmitted, in general:

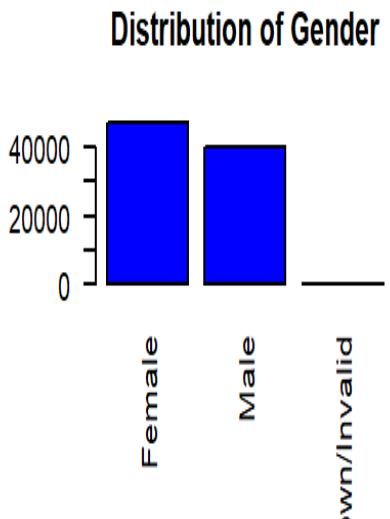
- Spend more time in the hospital
- Have more lab procedures done

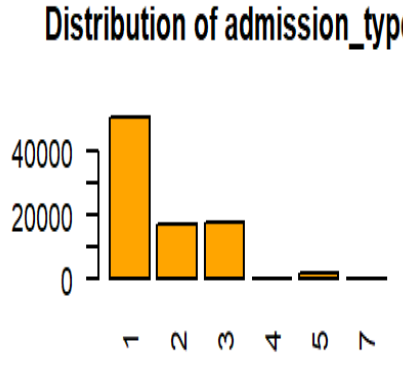
2.5 Distribution of the categorical variables

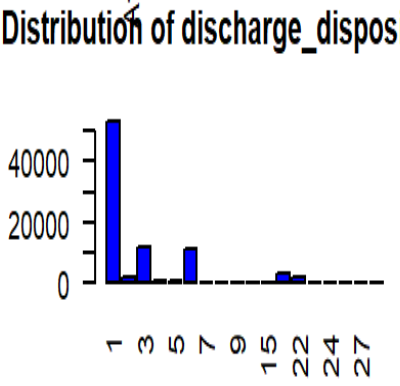
```
par(mfrow = c(3,3))

barplot(table(df$age), main="Distribution of Age", col="blue", las=2)
barplot(table(df$race), main="Distribution of Race", col="orange", las=2)
barplot(table(df$gender), main="Distribution of Gender", col="blue", las=2)

barplot(table(df$admission_type_id), main="Distribution of admission_type_id",
        col="orange", las=2)
barplot(table(df$discharge_disposition_id), main="Distribution of discharge_disposition_id",
        col="blue", las=2)
barplot(table(df$admission_source_id), main="Distribution of admission_source_id",
        col="orange",las=2)


barplot(table(df$primary_diagnosis), main="Distribution of primary_diagnosis",
        col="blue", las=2)
barplot(table(df$secondary_diagnosis), main="Distribution of secondary_diagnosis",
        col="orange", las=2)
barplot(table(df$secondary_diagnosis2), main="Distribution of secondary_diagnosis2",
        col="blue", las=2)
```

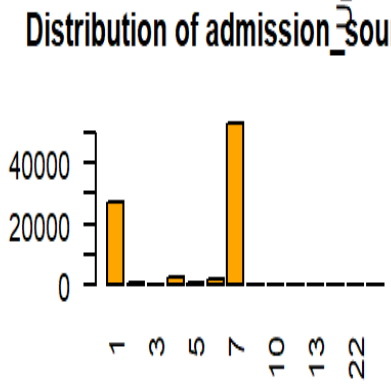

Distribution of Age
20000
10000
0
[0-10)
[10-20)
[20-30)
[30-40)
[40-50)
[50-60)
[60-70)
[70-80)
[80-90)
[90-100)

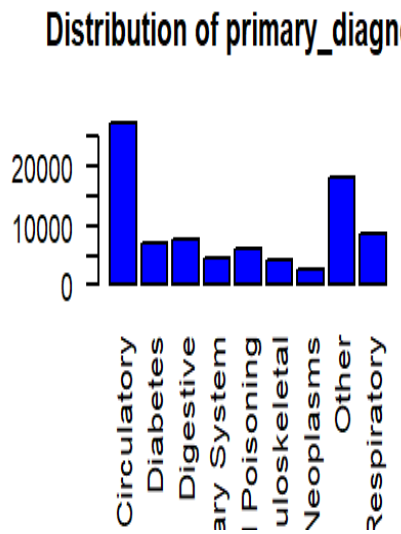

Distribution of Race
60000
40000
20000
0
AfricanAmerican
Asian
Caucasian
Hispanic
Other

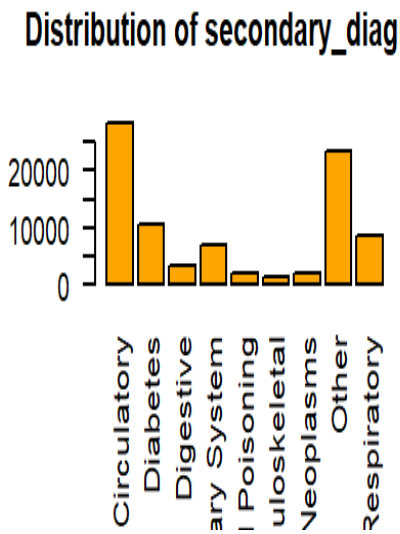

Distribution of Gender
40000
20000
0
Female
Male
Unknown/Invalid

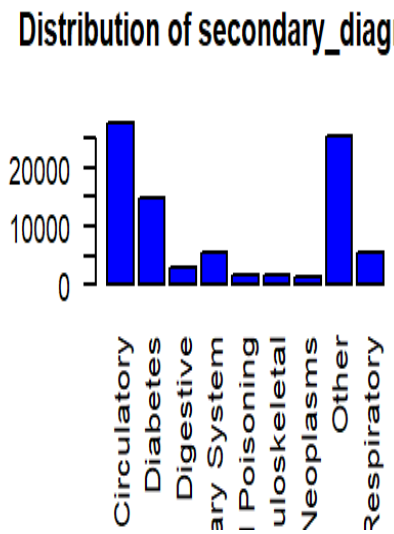

Distribution of admission_type_id
40000
20000
0
1
2
3
4
5
7


Distribution of discharge_disposition
40000
20000
0
1
3
5
7
9
15
22
24
27

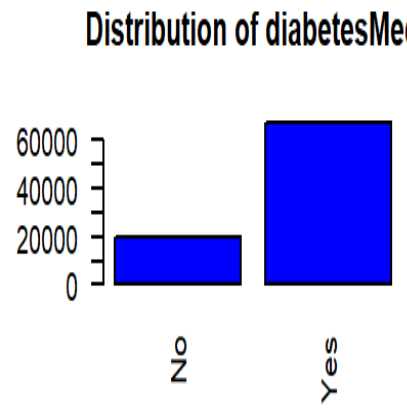

Distribution of admission_source_i
40000
20000
0
1
3
5
7
10
13
22

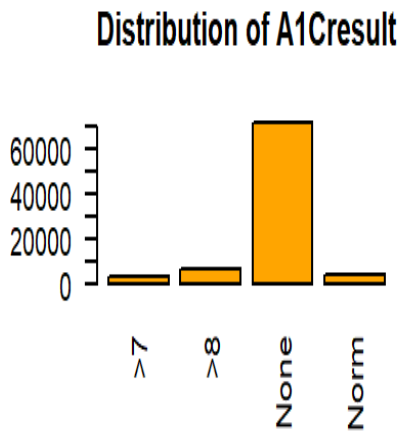

Distribution of primary_diagnosis
20000
10000
0
Circulatory
Diabetes
Digestive
Neoplasms
Other
Respiratory

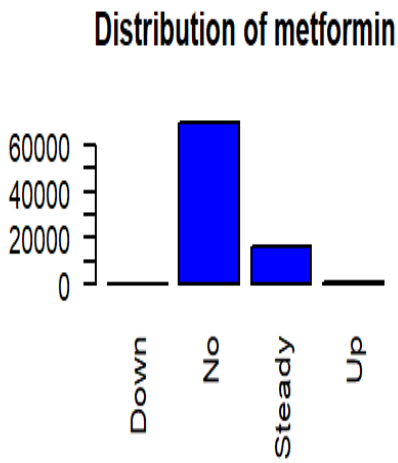

Distribution of secondary_diagnosi
20000
10000
0
Circulatory
Diabetes
Digestive
Neoplasms
Other
Respiratory

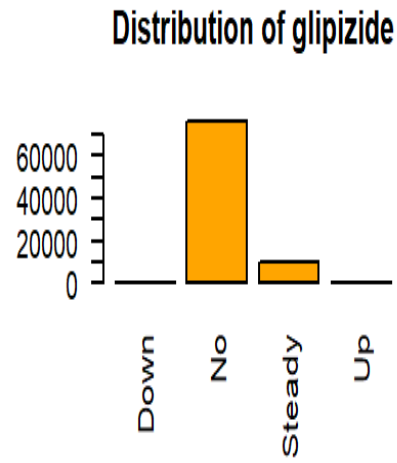

Distribution of secondary_diagnosis
20000
10000
0
Circulatory
Diabetes
Digestive
Neoplasms
Other
Respiratory

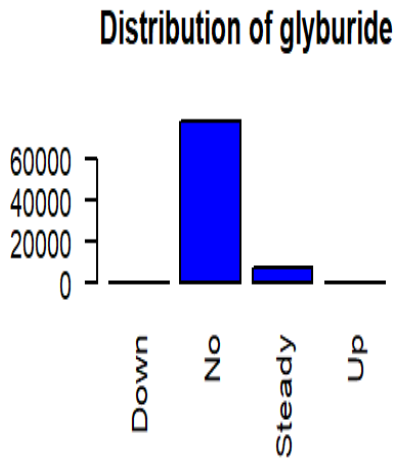
```
par(mfrow = c(3,3))

barplot(table(df$diabetesMed), main="Distribution of diabetesMed", col="blue", las=2)
barplot(table(df$A1Cresult), main="Distribution of A1Cresult", col="orange", las=2)
barplot(table(df$metformin), main="Distribution of metformin", col="blue", las=2)


barplot(table(df$glipizide), main="Distribution of glipizide", col="orange", las=2)
barplot(table(df$glyburide), main="Distribution of glyburide", col="blue", las=2)
barplot(table(df$pioglitazone), main="Distribution of pioglitazone", col="orange", las=2)


barplot(table(df$rosiglitazone), main="Distribution of rosiglitazone", col="blue", las=2)
barplot(table(df$insulin), main="Distribution of insulin", col="orange", las=2)
barplot(table(df$change), main="Distribution of change", col="blue", las=2)
```

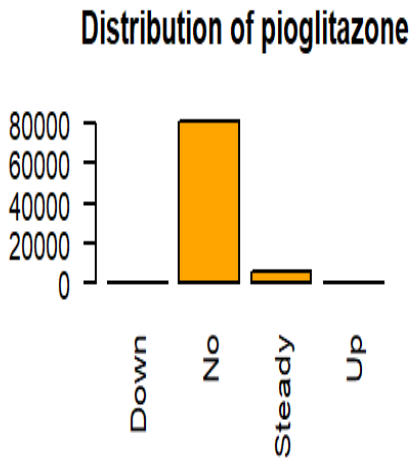


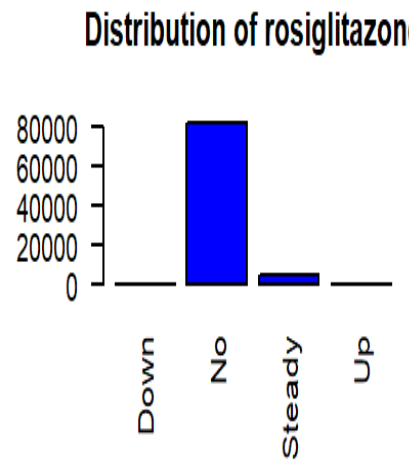


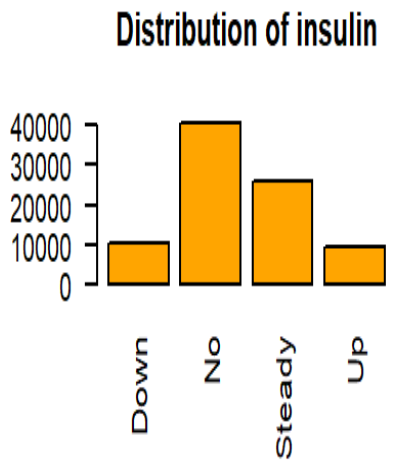


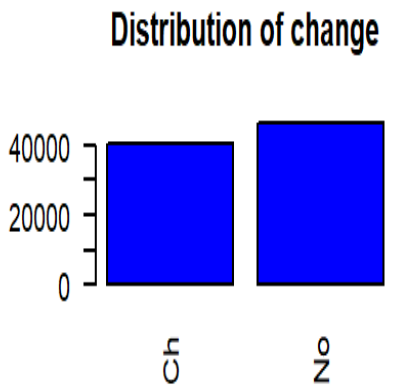








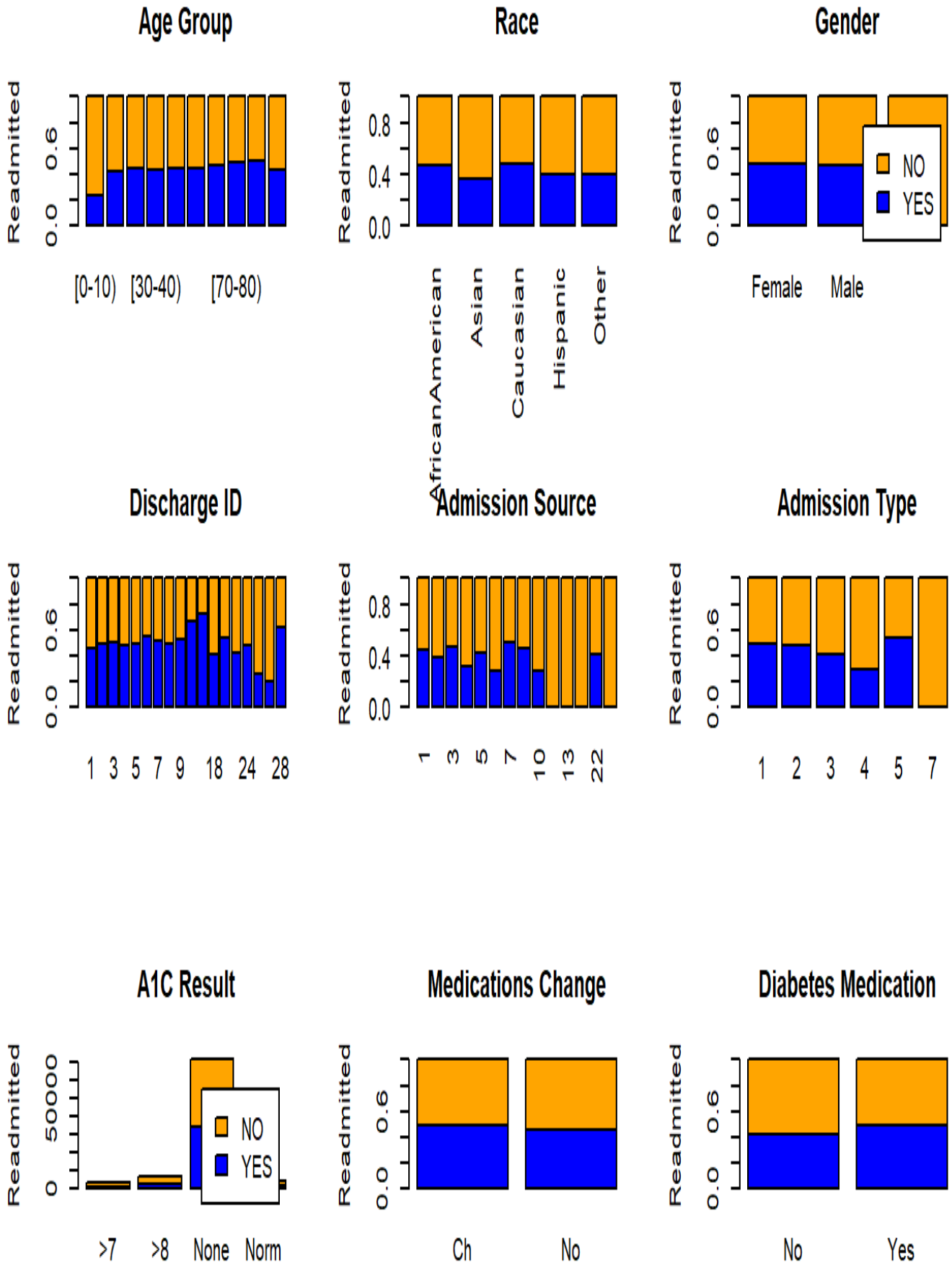




2.6 Explore the relationship between the categorical predictors and the predicted variable

```
# Frequency counts
ct_race <- table(df$readmitted, df$race)
ct_gender <- table(df$readmitted, df$gender)
ct_age <- table(df$readmitted, df$age)
ct_admissionType <- table(df$readmitted, df$admission_type_id)
ct_dischargeID <- table(df$readmitted, df$discharge_disposition_id)
ct_admissionSource <- table(df$readmitted, df$admission_source_id)
ct_A1C <- table(df$readmitted, df$A1Cresult)
ct_metf <- table(df$readmitted, df$metformin)
ct_glipz <- table(df$readmitted, df$glipizide)
ct_glyb <- table(df$readmitted, df$glyburide)
ct_piog <- table(df$readmitted, df$pioglitazone)
ct_rosig <- table(df$readmitted, df$rosiglitazone)
ct_insulin <- table(df$readmitted, df$insulin)
ct_change <- table(df$readmitted, df$change)
ct_diabMeds <- table(df$readmitted, df$diabetesMed)

# Create contingency tables showing proportions
ct_race1 <- prop.table(table(df$readmitted, df$race),2)
ct_gender1 <- prop.table(table(df$readmitted, df$gender),2)
ct_age1 <- prop.table(table(df$readmitted, df$age),2)
ct_admissionType1 <- prop.table(table(df$readmitted, df$admission_type_id),2)
ct_dischargeID1 <- prop.table(table(df$readmitted, df$discharge_disposition_id),2)
ct_admissionSource1 <- prop.table(table(df$readmitted, df$admission_source_id),2)
ct_A1C1 <- prop.table(table(df$readmitted, df$A1Cresult),2)
ct_metf1 <- prop.table(table(df$readmitted, df$metformin),2)
ct_glipz1 <- prop.table(table(df$readmitted, df$glipizide),2)
ct_glyb1 <- prop.table(table(df$readmitted, df$glyburide),2)
ct_piog1 <- prop.table(table(df$readmitted, df$pioglitazone),2)
ct_rosig1 <- prop.table(table(df$readmitted, df$rosiglitazone),2)
ct_insulin1 <- prop.table(table(df$readmitted, df$insulin),2)
ct_change1 <- prop.table(table(df$readmitted, df$change),2)
ct_diabMeds1 <- prop.table(table(df$readmitted, df$diabetesMed),2)
```

```
## Create barplots ##
# Patient Demographics (Legend removed for individual graphs where bars would be blocked)
par(mfrow=c(3,3))
barplot(ct_age1,main="Age Group", ylab="Readmitted",col=c("blue", "orange"))
barplot(ct_race1,main="Race", ylab="Readmitted",col=c("blue", "orange"),las=2)
barplot(ct_gender1,main="Gender", ylab="Readmitted",col=c("blue", "orange"),
        legend=rownames(ct_gender1))

barplot(ct_dischargeID1,main="Discharge ID", ylab="Readmitted",col=c("blue", "orange"))
barplot(ct_admissionSource1,main="Admission Source",
        ylab="Readmitted",col=c("blue", "orange"),las=2)
barplot(ct_admissionType1,main="Admission Type", ylab="Readmitted",col=c("blue", "orange"))



barplot(ct_A1C,main="A1C Result", ylab="Readmitted",col=c("blue", "orange"),legend=rownames(c
t_A1C))
barplot(ct_change1,main="Medications Change", ylab="Readmitted",col=c("blue", "orange"))
barplot(ct_diabMeds1,main="Diabetes Medication",ylab="Readmitted",col=c("blue", "orange"))
```



- Race: Asians were disproportionately less likely to be readmitted.

- Discharge ID: Discharge IDs 11, 19, and 20 were disproportionately less likely to be readmitted. Patients with discharge IDs 10,12,15 were disproportionately more likely to be readmitted.
- Admission sources 10, 11, 13, 14, 22, and 25 were less likely to be readmitted (0% readmission rates)
- Admission type 7 proportionately less likely to be readmitted
- No significant differences for the change in/prescription of drugs were observed.

3. Data Preprocessing

3.1 Select features

- From EDA, we observe that variables number_emergency, number_outpatient, number_inpatient, A1Cresult, glyburide, pioglitazone, pioglitazone, rosiglitazone, glipizide show low variance or are extremely skewed. They will be excluded from modeling.

```
df <- subset(df, select = -c(number_emergency, number_outpatient, number_inpatient, A1Cresul
t, glyburide, pioglitazone, pioglitazone, rosiglitazone, glipizide))

dim(df)
```

```
## [1] 86566    19
```

- After this step, 19 variables are left for modeling.

3.2 Remove outliers

- From visualizations in the EDA section, we observe some NUmerical variables including num_lab_procedures, number_diagnoses, and num_medications have outliers. We use 1.5 IQR to remove them.

```
# Remove outlier from column 'num_lab_procedures'
quartiles <- quantile(df$num_lab_procedures, probs=c(.25, .75), na.rm = FALSE)
IQR <- IQR(df$num_lab_procedures)
Lower <- quartiles[1] - 1.5*IQR
Upper <- quartiles[2] + 1.5*IQR

count_before <-dim(df)[1]
df<- subset(df, df$num_lab_procedures > Lower & df$num_lab_procedures < Upper)
count_after <-dim(df)[1]
cat(count_before-count_after, " outliers in num_lab_procedures have been removed. \n")
```

```
## 186  outliers in num_lab_procedures have been removed.
```

```
# Remove outliers from column 'number_diagnoses'
quartiles <- quantile(df$number_diagnoses, probs=c(.25, .75), na.rm = FALSE)
IQR <- IQR(df$number_diagnoses)
Lower <- quartiles[1] - 1.5*IQR
Upper <- quartiles[2] + 1.5*IQR

count_before <-dim(df)[1]
df <- subset(df, df$number_diagnoses > Lower & df$number_diagnoses < Upper)
count_after <-dim(df)[1]
cat(count_before-count_after, " outliers in number_diagnoses have been removed. \n")
```

```
## 52  outliers in number_diagnoses have been removed.
```

```
# Remove outliers from column 'num_medications'
quartiles <- quantile(df$num_medications, probs=c(.25, .75), na.rm = FALSE)
IQR <- IQR(df$num_medications)
Lower <- quartiles[1] - 2*IQR
Upper <- quartiles[2] + 2*IQR

count_before <-dim(df)[1]
df <- subset(df, df$num_medications > Lower & df$num_medications < Upper)
count_after <-dim(df)[1]
cat(count_before-count_after, " outliers in num_medications have been removed. \n")
```

```
## 1287  outliers in num_medications have been removed.
```

3.3 Partition data into training and test datasets using a 70% ratio

- readmitted is the predicted variable. It will be separated from the predictors.

```
#Separate X and Y using readmitted as predicted variable
dfX <- subset(df, select = -c(readmitted))
dfY <- subset(df, select = c(readmitted))

set.seed(123)
# Partition the dataset
trainRows <- createDataPartition(dfY$readmitted, p = .70, list = FALSE)

trainX <- dfX[trainRows,]
testX <-  dfX[-trainRows,]

trainY <- dfY[trainRows,]
testY <- dfY[-trainRows,]

dim(trainX)
```

```
## [1] 59529    18
```

```
dim(testX)
```

```
## [1] 25512    18
```

- After the partition, there are 59529 instances in the training set and 25512 instances in the test set.

3.4 Trandform and standardize features: center and scale

```
# center and scale the training set
train_tran <- preProcess(trainX, method=c("center", "scale"))
trainX <- predict(train_tran, trainX)

# center and scale the test set
testX <- predict(train_tran, testX)
```

3.5 Convert categorical variables to dummy variables

- Backup the training and testing sets before adding dummy variables for training different models

```
library(fastDummies)

# Some models do not require dummy variable, like the trees
# Will use the sets without dummies for models
trainX_noDummy <- trainX
testX_noDummy <- testX

colnames(trainX_noDummy)
```

```
##  [1] "race"                    "gender"
##  [3] "age"                     "admission_type_id"
##  [5] "discharge_disposition_id" "admission_source_id"
##  [7] "time_in_hospital"        "num_lab_procedures"
##  [9] "num_procedures"          "num_medications"
## [11] "number_diagnoses"        "metformin"
## [13] "insulin"                 "change"
## [15] "diabetesMed"             "primary_diagnosis"
## [17] "secondary_diagnosis"     "secondary_diagnosis2"
```

- Categorical variables will now be converted into n-1 dummy variables.

```
# Add dummy variables
trainX <- dummy_cols(trainX,
                     select_columns=c("race", "gender","age",
                                      "admission_type_id", "discharge_disposition_id","metfor
min",
                                      "insulin", "change","diabetesMed",
                                      "admission_source_id","primary_diagnosis",
                                      "secondary_diagnosis", "secondary_diagnosis2"),
                     remove_first_dummy=TRUE,
                     remove_selected_columns=TRUE)

testX <- dummy_cols(testX,
                     select_columns=c("race", "gender","age",
                                      "admission_type_id", "discharge_disposition_id","metfor
min",
                                      "insulin", "change","diabetesMed",
                                      "admission_source_id","primary_diagnosis",
                                      "secondary_diagnosis", "secondary_diagnosis2"),
                     remove_first_dummy=TRUE,
                     remove_selected_columns=TRUE)
```

3.6 PCA Analysis

- This is a high-dimensional dataset, we want to see if PCA can be applied to reduce feature dimension.

```
# Find the PCA from the training set
pca <- prcomp(trainX)
pca_var <- pca$sdev^2

# Find the percentage of each PCA component
pca_percents <- pca_var / sum(pca_var)
cat("Percentage of the first 15 PCAs: ", pca_percents[1:15], "\n")
```

```
## Percentage of the first 15 PCAs:  0.1910663 0.1092462 0.08477103 0.06680895 0.06017332 0.0
4253663 0.03169632 0.02533176 0.02446798 0.02444144 0.02292527 0.02127575 0.02083785 0.019348
05 0.01887004
```

```
# The total percentage of the first 35 PCAs
cat("The total percentage of the first 35 PCAs: ", sum(pca_percents[1:35]), "\n")
```

```
## The total percentage of the first 35 PCAs:  0.9428998
```

- We see there are no dominant PCA components. The first 35 components represent 94% of the features.

4. Modeling

4.0 Define functions

First we will define a function to calculate the metrics of a trained model

- The function will take one parameters: a trained model
- The function will calculate these metrics including accuracy, sensitivity, specificity, precision
- It will return a vector containing those metrics

```
# Define a function to calculate the metrics of a trained model
get_training_metrics <- function(model, roc) {

  # Total number of training objects
  total <- dim(trainX)[1]

 # Construct model's confusion matrix
  cm <- confusionMatrix(model, norm = "none")

  # Calculate metrics
  accuracy <- round((cm$table[1,1] + cm$table[2,2]) / total, 3)
  sensitivity <- round(cm$table[1,1] / (cm$table[1,1] + cm$table[2,1]), 3)
  specificity <- round(cm$table[2,2] / (cm$table[2,2] + cm$table[1,2]), 3)
  precision <- round(cm$table[1,1] / (cm$table[1,1] + cm$table[1,2]), 3)
  recall <- sensitivity
  F1 <- round(2 * recall * precision / (recall + precision ), 3)

  # Return a vector of metrics
  c(accuracy, sensitivity, specificity, precision, recall, F1, round(roc,3))
}
```

Next we will define a function to calculate the metrics of prediction result on test dataset

- The function will take one parameters: predicted results of the test set
- The function will calculate these metrics including ROC, accuracy, sensitivity, specificity, precision
- It will return a vector containing those metrics

```
# Define a function to calculate the metrics of a trained model
get_test_metrics <- function(test_results) {
  total <- dim(testX)[1]

  # Construct prediction's confusion matrix
  cm <- confusionMatrix(test_results, testY, positive = "YES")

  # Calculate metrics
  accuracy <- round((cm$table[1,1] + cm$table[2,2]) / total, 3)
  sensitivity <- round(cm$table[1,1] / (cm$table[1,1] + cm$table[2,1]), 3)
  specificity <- round(cm$table[2,2] / (cm$table[2,2] + cm$table[1,2]), 3)
  precision <- round(cm$table[1,1] / (cm$table[1,1] + cm$table[1,2]), 3)
  recall <- sensitivity
  F1 <- round(2 * recall * precision / (recall + precision ), 3)
  # Return a vector of metrics
  c(accuracy, sensitivity, specificity, precision, recall, F1)
}
```

4.1 Logistic Regression Model

- First we will define a control variable that will be used to train all the models.
- We will use a 10 fold cross-validation method to optimize the training process.
- For each model, we will first train the model using training dataset, then use test dataset to test the model.
- We will generate a data frame to hold the training and testing performance metrics.

```
# Define train control
ctrl <- trainControl(method = "cv", summaryFunction = twoClassSummary,
                     classProbs = TRUE, savePredictions = "final")

# Logistic Regression
set.seed(123)
lrFit <- train(x = trainX, y = trainY,
               method = "glm", metric = "ROC", trControl = ctrl)

# Calculate training/resampling performance metrics
metrics_tr <- data.frame(Metric.Train = c("Accuracy", "Sensitivity", "Specificity", "Precisio
n", "Recall", "F-Measure", "ROC"))

metrics_tr$LR <- get_training_metrics(lrFit, lrFit$results$ROC)

#metrics_tr

# Predict on test data
lrTestResults <- predict(lrFit, testX)

# Calculate test performance metrics
metrics_test <- data.frame(Metric.Test = c("Accuracy", "Sensitivity", "Specificity", "Precisi
on", "Recall", "F-Measure"))
metrics_test$LR <- get_test_metrics(lrTestResults)

# Importance of the predictors
lrImp <- varImp(lrFit, scale = FALSE)

# Display model's performance
metrics_tr[, c("Metric.Train", "LR")]
```

```
##   Metric.Train    LR
## 1     Accuracy 0.585
## 2  Sensitivity 0.524
## 3  Specificity 0.640
## 4    Precision 0.570
## 5       Recall 0.524
## 6    F-Measure 0.546
## 7          ROC 0.617
```

```
metrics_test[, c("Metric.Test", "LR")]
```

```
##   Metric.Test    LR
## 1    Accuracy 0.586
## 2 Sensitivity 0.524
## 3 Specificity 0.643
## 4   Precision 0.572
## 5      Recall 0.524
## 6   F-Measure 0.547
```

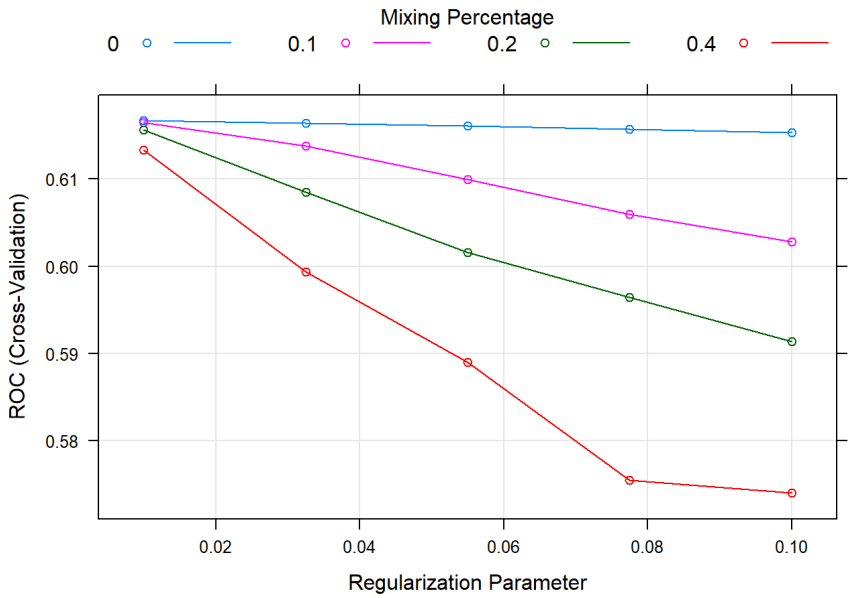
- The training (cross validation) performance and testing performance for logistic regression model is displayed in the above metrics.

4.2 Penalized Logistic Regression Model

For penalized logistic regression model, we use the following perimeters to build the tuning grid:

- alpha values: 0, 0,0.1,0.2,0.4
- regularization perimeter lambda takes 5 values evenly between 0.01 and 0.1
- length: 5

```
# Penalized Logistic Regression
set.seed(123)
glmnGrid <- expand.grid(alpha=c(0,0.1,0.2,0.4),
                        lambda=seq(.01, .1, length=5))
glmnFit <- train(x = trainX, y = trainY,
                 method="glmnet", tuneGrid=glmnGrid,
                 metric="ROC", trControl=ctrl)
#glmnFit

# Calculate training/resampling performance metrics
metrics_tr$GLMN <- get_training_metrics(glmnFit, glmnFit$results$ROC[1])
#metrics_tr

# Predict on test data
glmnTestResults <- predict(glmnFit, testX)

# Calculate test performance metrics
metrics_test$GLMN <- get_test_metrics(glmnTestResults)

# Importance of the predictors
glmnImp <- varImp(glmnFit, scale = FALSE)


# Plot the tuning results
plot(glmnFit)
```




```
# Display model's performance
metrics_tr[, c("Metric.Train", "GLMN")]
```

```
##   Metric.Train  GLMN
## 1     Accuracy 0.585
## 2  Sensitivity 0.521
## 3  Specificity 0.642
## 4    Precision 0.570
## 5       Recall 0.521
## 6    F-Measure 0.544
## 7          ROC 0.617
```

```
metrics_test[, c("Metric.Test", "GLMN")]
```

```
##   Metric.Test  GLMN
## 1    Accuracy 0.586
## 2 Sensitivity 0.520
## 3 Specificity 0.646
## 4   Precision 0.572
## 5      Recall 0.520
## 6   F-Measure 0.545
```

- The cross validation result shows the optimal model comes when alpha=0, lambda=0.01
- The training (cross validation) performance and testing performance is displayed in the above metrics.

4.3 Nearest Shrunken Centroids Model

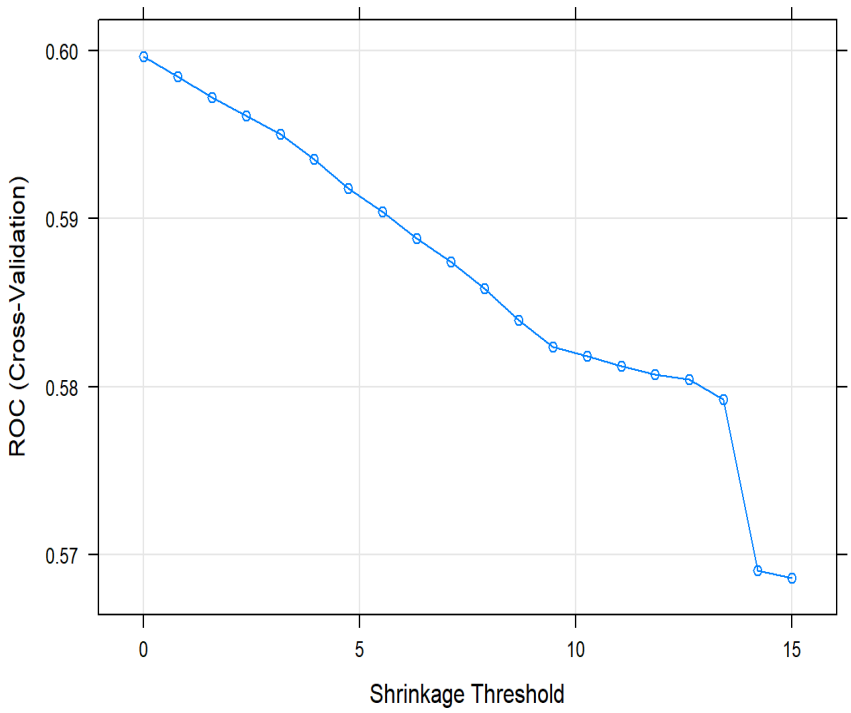
For Nearest Shrunken Centroids model, we use the following perimeter to build the tuning grid:

- threshold takes 20 values evenly spaced between 0 and 15

```
# Nearest Shrunken Centroids
set.seed(123)
nscFit <- train(x=trainX,y=trainY,
                method="pam", tuneGrid=data.frame(threshold=seq(0,15, length=20)),
                metric="ROC", trControl=ctrl)
```

```
## 11111111111
```

```
#nscFit

# Calculate training/resampling performance metrics
metrics_tr$NSC <- get_training_metrics(nscFit, nscFit$results$ROC[1])

# Predict on test data
nscTestResults <- predict(nscFit, testX)

# Calculate test performance metrics
metrics_test$NSC <- get_test_metrics(nscTestResults)

# Importance of the predictors
nscImp <- varImp(nscFit, scale = FALSE)

# Plot the tuning result
plot(nscFit)
```




```
# Display model's performance
metrics_tr[, c("Metric.Train", "NSC")]
```

```
##   Metric.Train   NSC
## 1     Accuracy 0.570
## 2  Sensitivity 0.504
## 3  Specificity 0.629
## 4    Precision 0.553
## 5       Recall 0.504
## 6    F-Measure 0.527
## 7          ROC 0.600
```

```
metrics_test[, c("Metric.Test", "NSC")]
```

```
##   Metric.Test   NSC
## 1    Accuracy 0.575
## 2 Sensitivity 0.507
## 3 Specificity 0.637
## 4   Precision 0.559
## 5      Recall 0.507
## 6   F-Measure 0.532
```

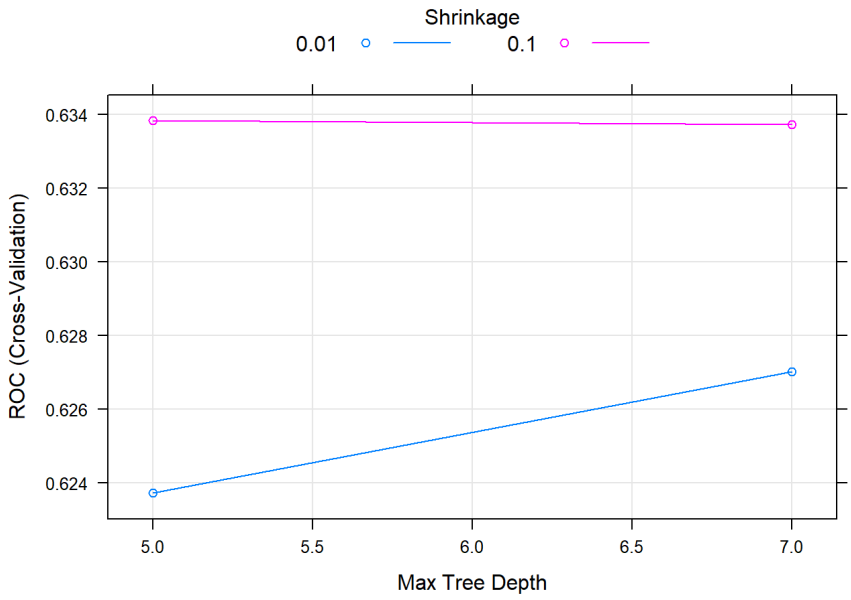
- The cross validation result shows the optimal model comes when shrinkage threshhold is 0.
- The training (cross validation) performance and testing performance is displayed in the above metrics.

4.4 Boosted Trees

For the boosted trees model, we use the following perimeter to build the tuning grid:

- interaction depth: 5, 7
- number of trees: 500
- shrinkage: 0.01, 0.1

```
gbmGrid <- expand.grid(interaction.depth = c(5, 7),
                       n.trees = 500,
                       shrinkage = c(.01, .1),
                       n.minobsinnode = 5)

set.seed(123)

gbmFit <- train(x = trainX_noDummy, y = trainY,
                method = "gbm", tuneGrid = gbmGrid,
                verbose = FALSE, metric = "ROC", trControl = ctrl)

# Calculate training/resampling performance metrics
metrics_tr$GBM <- get_training_metrics(gbmFit, gbmFit$results$ROC[3])
#metrics_tr

# Predict on test data
gbmTestResults <- predict(gbmFit, testX_noDummy)

# Calculate test performance metrics
metrics_test$GBM <- get_test_metrics(gbmTestResults)

#The tunning result
#gbmFit
plot(gbmFit)
```




```
# Display model's performance
metrics_tr[, c("Metric.Train", "GBM")]
```

```
##   Metric.Train   GBM
## 1     Accuracy 0.596
## 2  Sensitivity 0.561
## 3  Specificity 0.628
## 4    Precision 0.578
## 5       Recall 0.561
## 6    F-Measure 0.569
## 7          ROC 0.627
```

```
metrics_test[, c("Metric.Test", "GBM")]
```

```
##   Metric.Test   GBM
## 1    Accuracy 0.601
## 2 Sensitivity 0.565
## 3 Specificity 0.633
## 4   Precision 0.583
## 5      Recall 0.565
## 6   F-Measure 0.574
```

- The cross validation result shows the optimal model comes when shrinkage is 0.1 and depth is 5..
- The training (cross validation) performance and testing performance is displayed in the above metrics.

4.5 Bagged Tree

For bagged tree model, we use the following perimeter to train:

- number of bag(nbagg) = 30

```
set.seed(123)

trbagFit <- train(x = trainX_noDummy, y = trainY,
                  method = "treebag",
                  nbagg = 30,
                  metric = "ROC",
                  trControl = ctrl)

#trbagFit

# Calculate training/resampling performance metrics
metrics_tr$TRBAG <- get_training_metrics(trbagFit, trbagFit$results$ROC)

# Predict on test data
trbagTestResults <- predict(trbagFit, testX_noDummy)

# Calculate test performance metrics
metrics_test$TRBAG <- get_test_metrics(trbagTestResults)

# Display model's performance
metrics_tr[, c("Metric.Train", "TRBAG")]
```

```
##   Metric.Train TRBAG
## 1     Accuracy 0.572
## 2  Sensitivity 0.541
## 3  Specificity 0.601
## 4    Precision 0.552
## 5       Recall 0.541
## 6    F-Measure 0.546
## 7          ROC 0.603
```

```
metrics_test[, c("Metric.Test", "TRBAG")]
```

```
##   Metric.Test TRBAG
## 1    Accuracy 0.569
## 2 Sensitivity 0.535
## 3 Specificity 0.601
## 4   Precision 0.549
## 5      Recall 0.535
## 6   F-Measure 0.542
```

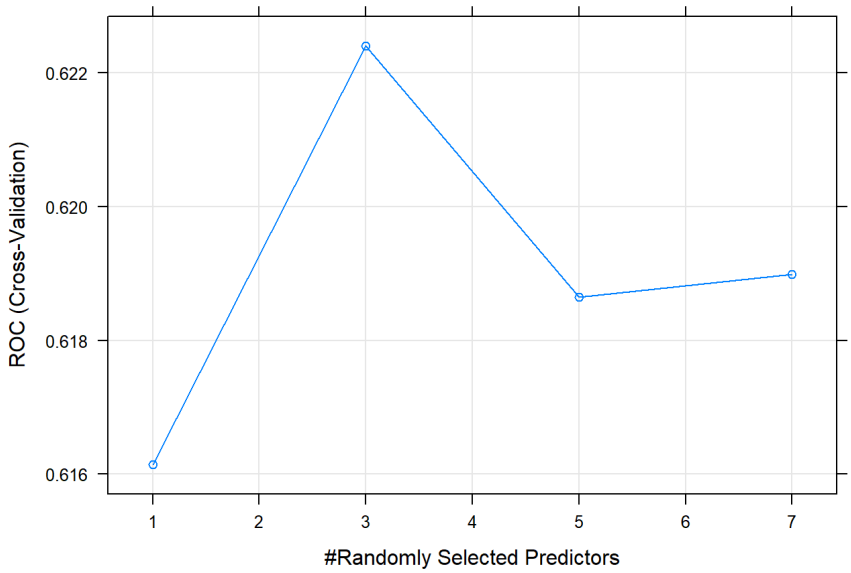
- The training (cross validation) performance and testing performance is displayed in the above metrics.

4.6 Random Forest Tree

For random forest tree model, we use the following perimeter to build the tuning grid:

- number of randomly selected predictors: 1, 3, 5, 7
- number of trees: 100

```
mtryValues <- seq(1,8,2)

set.seed(123)
rfFit <- train(x = trainX_noDummy, y = trainY,
                method = "rf",
                ntree = 100,
                tuneGrid = data.frame(mtry = mtryValues),
                metric = "ROC",
                trControl = ctrl)


# Calculate training/resampling performance metrics
metrics_tr$RF <- get_training_metrics(rfFit, rfFit$results$ROC[2])

# Predict on test data
rfTestResults <- predict(rfFit, testX_noDummy)

# Calculate test performance metrics
metrics_test$RF <- get_test_metrics(rfTestResults)

# Importance of the predictors
rfImp <- varImp(rfFit, scale = FALSE)

# Plot tuning result
#rfFit
plot(rfFit)
```




```
# Display model's performance
metrics_tr[, c("Metric.Train", "RF")]
```

```
##   Metric.Train    RF
## 1     Accuracy 0.586
## 2  Sensitivity 0.532
## 3  Specificity 0.635
## 4    Precision 0.570
## 5       Recall 0.532
## 6    F-Measure 0.550
## 7          ROC 0.622
```

```
metrics_test[, c("Metric.Test", "RF")]
```

```
##   Metric.Test    RF
## 1    Accuracy 0.587
## 2 Sensitivity 0.503
## 3 Specificity 0.663
## 4   Precision 0.576
## 5      Recall 0.503
## 6   F-Measure 0.537
```

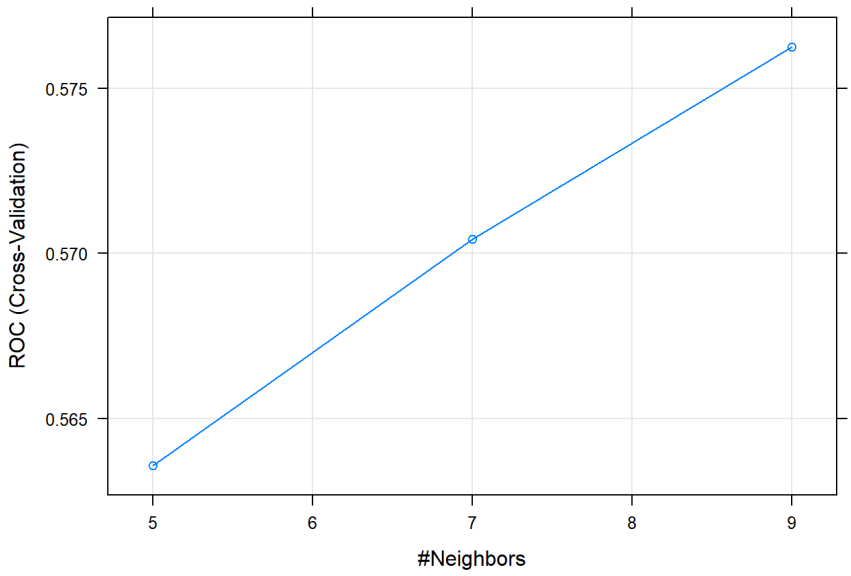
- The cross validation result shows the optimal model comes when the number of randomly selected predictors is 3
- The training (cross validation) performance and testing performance is displayed in the above metrics.

4.7 K-Nearest Neighbor (KNN) Model

For KNN model, we use the following perimeter to build the tuning grid:

- tuneLength = 3
- number of neighbors: 5, 7, 9

```
set.seed(123)

knnFit <- train(x = trainX, y = trainY,
                method = "knn",
                tuneLength = 3,
                metric = "ROC", trControl = ctrl)


# Calculate training/resampling performance metrics
metrics_tr$KNN <- get_training_metrics(knnFit, knnFit$results$ROC[3])

# Predict on test data
knnTestResults <- predict(knnFit, testX)

# Calculate test performance metrics
metrics_test$KNN <- get_test_metrics(knnTestResults)

# Importance of the predictors
knnImp <- varImp(knnFit, scale = FALSE)

# Plot the tuning results
#knnFit
plot(knnFit)
```




```
# Display model's performance
metrics_tr[, c("Metric.Train", "KNN")]
```

```
##   Metric.Train   KNN
## 1     Accuracy 0.555
## 2  Sensitivity 0.527
## 3  Specificity 0.581
## 4    Precision 0.534
## 5       Recall 0.527
## 6    F-Measure 0.530
## 7          ROC 0.576
```

```
metrics_test[, c("Metric.Test", "KNN")]
```

```
##   Metric.Test   KNN
## 1    Accuracy 0.557
## 2 Sensitivity 0.528
## 3 Specificity 0.583
## 4   Precision 0.535
## 5      Recall 0.528
## 6   F-Measure 0.531
```

- The cross validation result shows the optimal model comes when K is 9.
- The training (cross validation) performance and testing performance is displayed in the above metrics.

5. Model Evaluation and Conclusion

5.1 Baseline Model

- For this data analysis, a model's ability to predict the positive (readmitted-Yes) accurately is the most important metric. Therefore, we choose All Positive Model as the base model.

```
round(table(trainY) / length(trainY), 3)
```

```
## trainY
##   YES    NO
## 0.476 0.524
```

- From the above table, we see that when we assign all predictions as positive, the accuracy of this base model is 0.476

5.2 Calculate AUC (Area Under ROC Curve)

For each model, based on the trained models' result, we first generate ROC, then calculate the area under ROC curve (AUC)

```
#ROC Curve
library(pROC)
lrROC <- roc(response=trainY,predictor=lrFit$pred$YES,levels=rev(levels(lrFit$pred$obs)))
glmnROC <- roc(response=trainY,predictor=glmnFit$pred$YES,levels=rev(levels(glmnFit$pred$ob
s)))
nscROC <- roc(response=trainY,predictor=nscFit$pred$YES,levels=rev(levels(nscFit$pred$obs)))
gbmROC <- roc(response=trainY,predictor=gbmFit$pred$YES,levels=rev(levels(gbmFit$pred$obs)))
rfROC <- roc(response=trainY,predictor=rfFit$pred$YES,levels=rev(levels(rfFit$pred$obs)))
trbagROC <- roc(response=trainY,predictor=trbagFit$pred$YES,levels=rev(levels(trbagFit$pred$o
bs)))
knnROC <- roc(response=trainY,predictor=knnFit$pred$YES,levels=rev(levels(knnFit$pred$obs)))

lrAUC <-round(auc(lrROC), 3)
glmnAUC <-round(auc(glmnROC), 3)
nscAUC <-round(auc(nscROC), 3)
gbmAUC <-round(auc(gbmROC), 3)
rfAUC <-round(auc(rfROC), 3)
trbagAUC <-round(auc(trbagROC), 3)
knnAUC <-round(auc(knnROC), 3)

# Get Area under the ROC
metrics_tr <- rbind(metrics_tr, c("AUC", lrAUC, glmnAUC, nscAUC,
                                  gbmAUC, rfAUC, trbagAUC, knnAUC))
```

- The AUC values are added to the training model's performance metrics data frame.

5.3 Compare models' performance metrics

- Compare Models' training (cross-validation) performance

```
metrics_tr
```

```
##   Metric.Train    LR  GLMN   NSC   GBM TRBAG    RF   KNN
## 1     Accuracy 0.585 0.585  0.57 0.596 0.572 0.586 0.555
## 2  Sensitivity 0.524 0.521 0.504 0.561 0.541 0.532 0.527
## 3  Specificity  0.64 0.642 0.629 0.628 0.601 0.635 0.581
## 4    Precision  0.57  0.57 0.553 0.578 0.552  0.57 0.534
## 5       Recall 0.524 0.521 0.504 0.561 0.541 0.532 0.527
## 6    F-Measure 0.546 0.544 0.527 0.569 0.546  0.55  0.53
## 7          ROC 0.617 0.617   0.6 0.627 0.603 0.622 0.576
## 8          AUC 0.503  0.51 0.502 0.502 0.499 0.503 0.495
```

- Compare Models' test performance

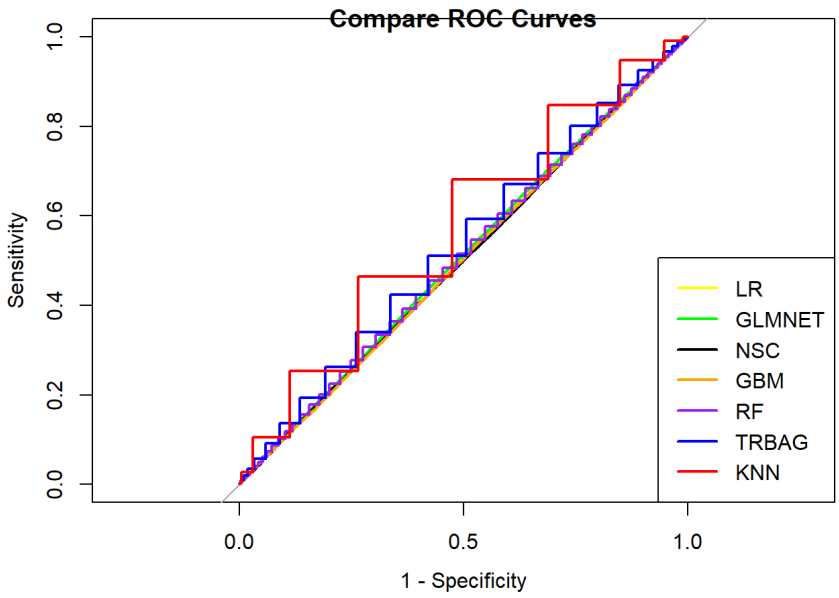
```
metrics_test
```

```
##   Metric.Test    LR  GLMN   NSC   GBM TRBAG    RF   KNN
## 1    Accuracy 0.586 0.586 0.575 0.601 0.569 0.587 0.557
## 2 Sensitivity 0.524 0.520 0.507 0.565 0.535 0.503 0.528
## 3 Specificity 0.643 0.646 0.637 0.633 0.601 0.663 0.583
## 4   Precision 0.572 0.572 0.559 0.583 0.549 0.576 0.535
## 5      Recall 0.524 0.520 0.507 0.565 0.535 0.503 0.528
## 6   F-Measure 0.547 0.545 0.532 0.574 0.542 0.537 0.531
```

5.4 Plot Roc Curves

- We plot the ROC curves for all trainings models in one graph. The comparison can be seen with different marking colors.

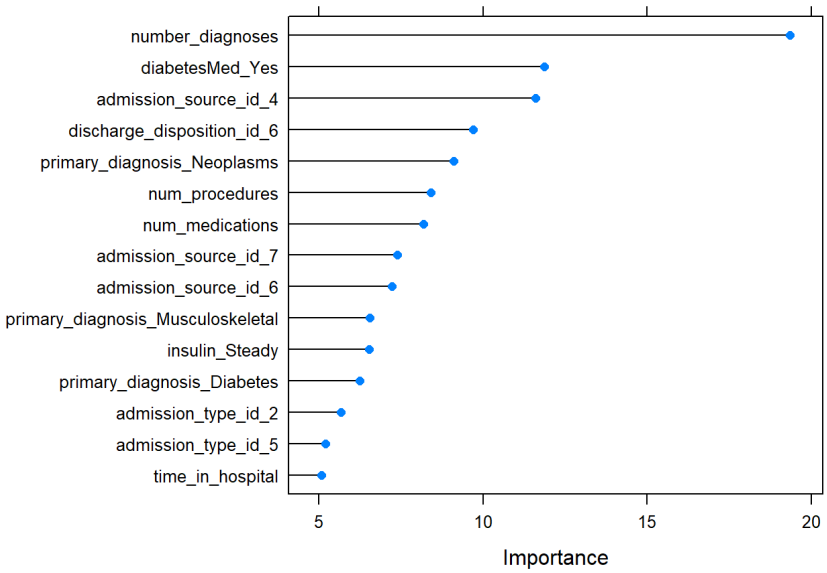
```
plot(lrROC, type="s", col='yellow', legacy.axes=TRUE)
plot(glmnROC, type="s", add=TRUE, col='green', legacy.axes=TRUE)
plot(nscROC, type="s", add=TRUE, col='black', legacy.axes=TRUE)
plot(gbmROC, type="s", add=TRUE, col='orange',legacy.axes=TRUE)
plot(rfROC, type="s", add=TRUE, col='purple', legacy.axes=TRUE)
plot(trbagROC, type="s", add=TRUE, col='blue',legacy.axes=TRUE)
plot(knnROC, type="s", add=TRUE, col='red', legacy.axes=TRUE)

legend("bottomright", legend=c("LR", "GLMNET", "NSC", "GBM", "RF", "TRBAG", "KNN"),
       col=c("yellow","green", "black",'orange', 'purple','blue', 'red'), lwd=2)


title(main="Compare ROC Curves")
```



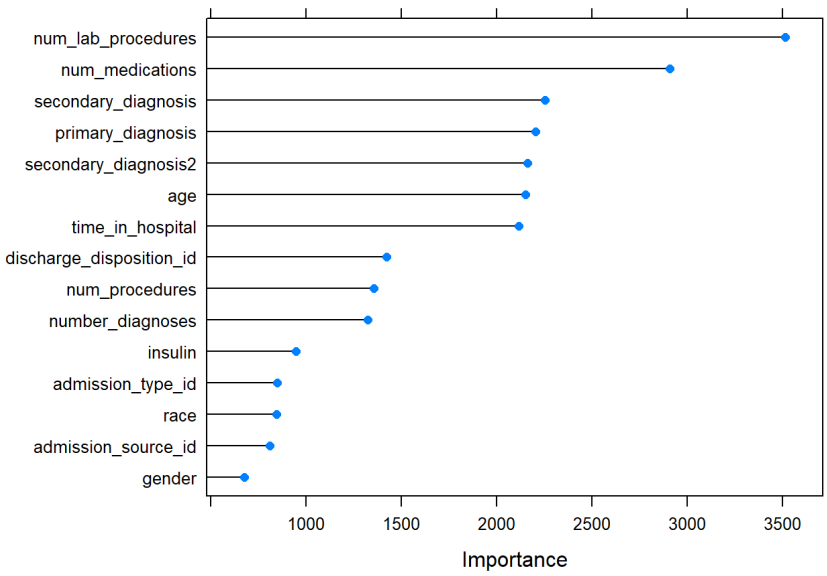
5.5 Check Feature's Importance

- We displayed the first 15 important features from logistic regression model, random forest model, and KNN model respectively.

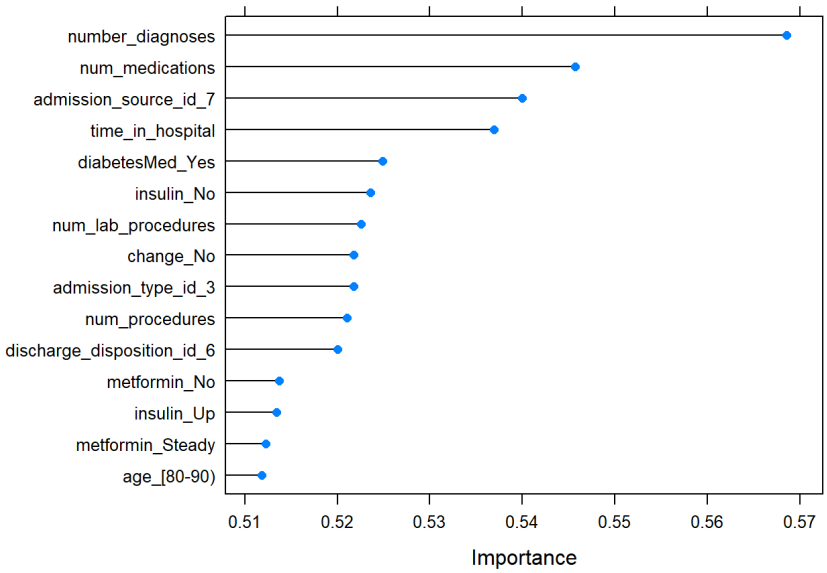
```
plot(lrImp, top = 15)
```




```
plot(rfImp, top = 15)
```




```
plot(knnImp, top = 15)
```



- Though each model displays the important features in different order, some features appear commonly in the top list.
- These important features include: number_diagnoses, num_procedures, num_lab_procedures, num_medication, time_in_hospital. etc.