Methods of Data Science IS 517: Final Project
Stroke Prediction based on health and lifestyle factors

Saish Desai, Anish Shetty, Vishnupriya Singh

1. Importing all the libraries

2. Introduction

A stroke occurs when the oxygen and nutrient supply to the brain is interrupted by either a blocked artery or the leaking/bursting of a blood vessel. When that happens, the brain cannot get the blood and oxygen it needs, due to which the brain cells die. According to the American Stroke Organization, Stroke is the No 5. Cause of death and the leading cause of disability in the United States, and may depend on a person's health, habits, and lifestyle. For this project, we have taken a data set from Kaggle with predictors representing all these causes of stroke with an aim to predict whether a person is likely to get a stroke. The data set has '5110' observations, where each observation corresponds to a single person. The data set has 12 variables – 11 independent variables/predictors and 1 binary response variable. The outcome is '1' if the person gets a stroke and '0' otherwise.

3. Research Questions

1) Given the information of a person from the list of predictors, can we predict if the person is likely to get a stroke?
2) Which factors are important in influencing whether a person is likely to get a stroke or not?
3) How do we tackle the problem of an imbalanced dataset with positive class (stroke) as a minority?

3. Data Cleaning & Analysis

Dataset Source : https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset

```
stroke_data = read.csv("healthcare-dataset-stroke-data.csv")
head(stroke_data)
```

```
##      id gender age hypertension heart_disease ever_married     work_type
## 1  9046   Male  67            0             1          Yes       Private
## 2 51676 Female  61            0             0          Yes Self-employed
## 3 31112   Male  80            0             1          Yes       Private
## 4 60182 Female  49            0             0          Yes       Private
## 5  1665 Female  79            1             0          Yes Self-employed
## 6 56669   Male  81            0             0          Yes       Private
##   Residence_type avg_glucose_level  bmi  smoking_status stroke
## 1          Urban            228.69 36.6 formerly smoked      1
## 2          Rural            202.21  N/A    never smoked      1
## 3          Rural            105.92 32.5    never smoked      1
## 4          Urban            171.23 34.4          smokes      1
```

```
## 5          Rural            174.12   24    never smoked      1
## 6          Urban            186.21   29 formerly smoked      1
```

Dataset Description and EDA

After describing the data we observed that there is a third category of gender and it has only one entry. In our analysis we are considering only Males and Females, hence we will remove this single entry.

```
str(stroke_data)
```

```
## 'data.frame':    5110 obs. of  12 variables:
##  $ id               : int  9046 51676 31112 60182 1665 56669 53882 10434 27419 60491 ...
##  $ gender           : chr  "Male" "Female" "Male" "Female" ...
##  $ age              : num  67 61 80 49 79 81 74 69 59 78 ...
##  $ hypertension     : int  0 0 0 0 1 0 1 0 0 0 ...
##  $ heart_disease    : int  1 0 1 0 0 0 1 0 0 0 ...
##  $ ever_married     : chr  "Yes" "Yes" "Yes" "Yes" ...
##  $ work_type        : chr  "Private" "Self-employed" "Private" "Private" ...
##  $ Residence_type   : chr  "Urban" "Rural" "Rural" "Urban" ...
##  $ avg_glucose_level: num  229 202 106 171 174 ...
##  $ bmi              : chr  "36.6" "N/A" "32.5" "34.4" ...
##  $ smoking_status   : chr  "formerly smoked" "never smoked" "never smoked" "smokes" ...
##  $ stroke           : int  1 1 1 1 1 1 1 1 1 1 ...
```

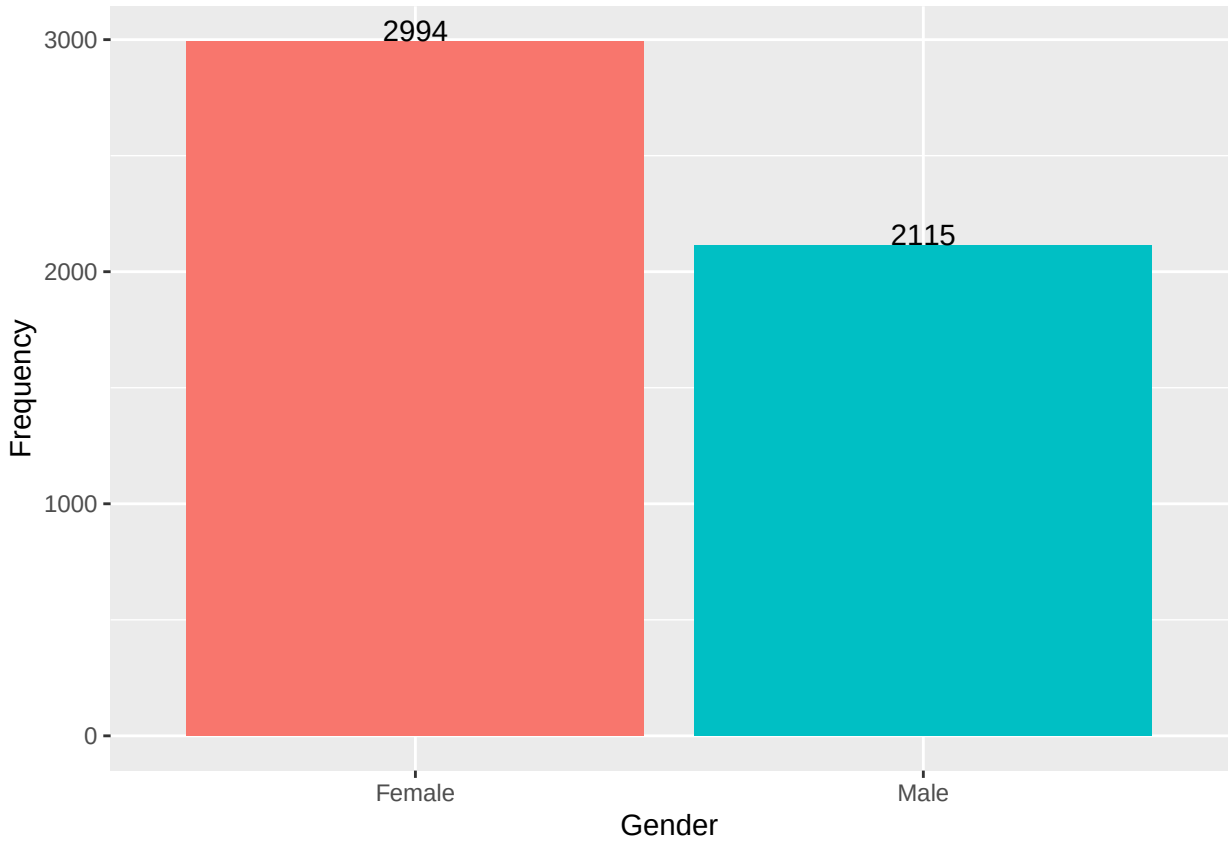
```
# Remvoing gender entries apart from 'Male' and 'Female'
stroke_data <- stroke_data[stroke_data$gender != 'Other',]
```

Understanding the distribution of variables.

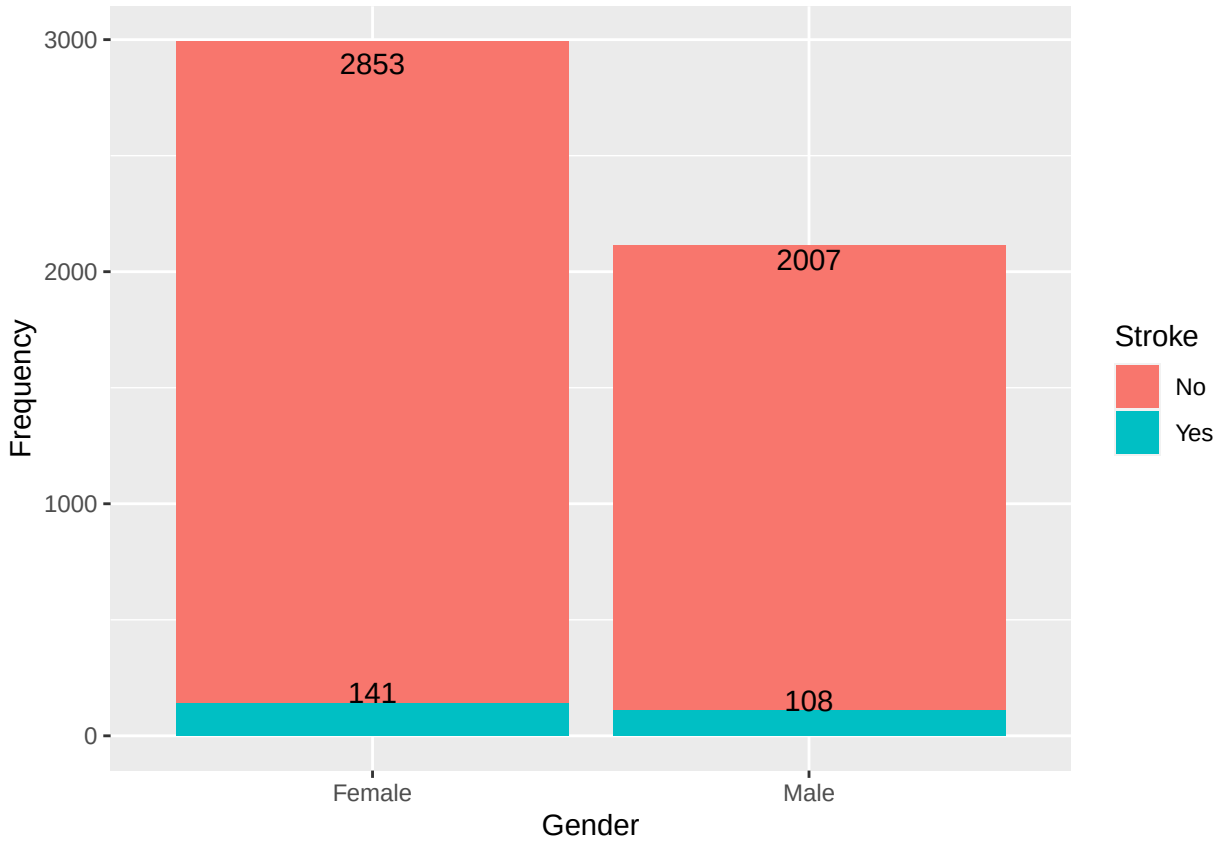
Categorical Variables

1. Gender

```
gender_counts <- as.data.frame(table(stroke_data$gender))

ggplot(gender_counts, aes(x = Var1, y = Freq, fill = Var1)) +
        geom_bar(stat = "identity") + theme(legend.position="none") +
        geom_text(aes(label = Freq), vjust = 0) +
        labs(x ="Gender", y = "Frequency")
```

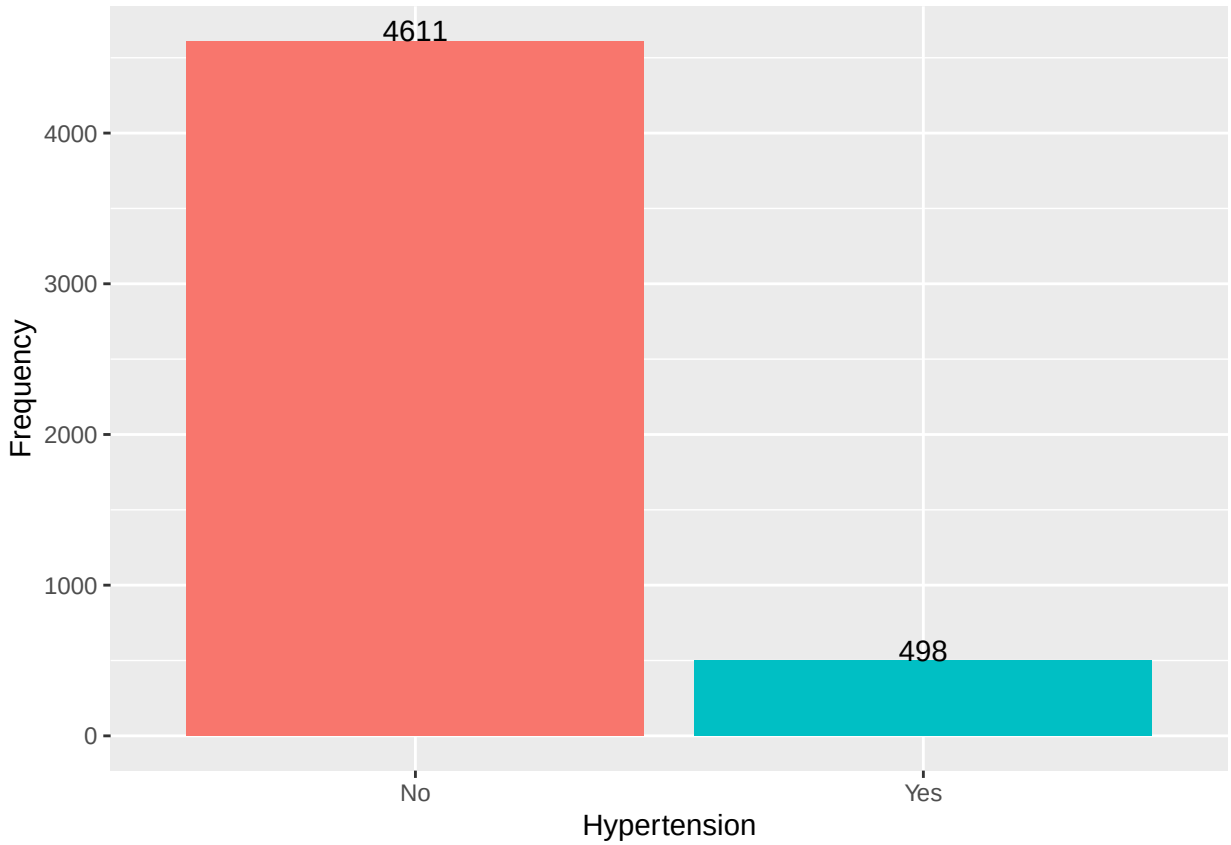



```
gender_counts <- as.data.frame(table(stroke_data$gender, stroke_data$stroke))
gender_counts$Var1 <- if_else(gender_counts$Var1 == "Female", 'Female','Male')
gender_counts$Var2 <- if_else(gender_counts$Var2 == 0, 'No','Yes')

gender_plot <- ggplot(gender_counts, aes(x = Var1, y = Freq, fill = Var2)) +
        geom_bar(stat = "identity")  +
        geom_text(aes(label = Freq), vjust = 0) +
        labs(x ="Gender", y = "Frequency", fill = 'Stroke')
gender_plot
```

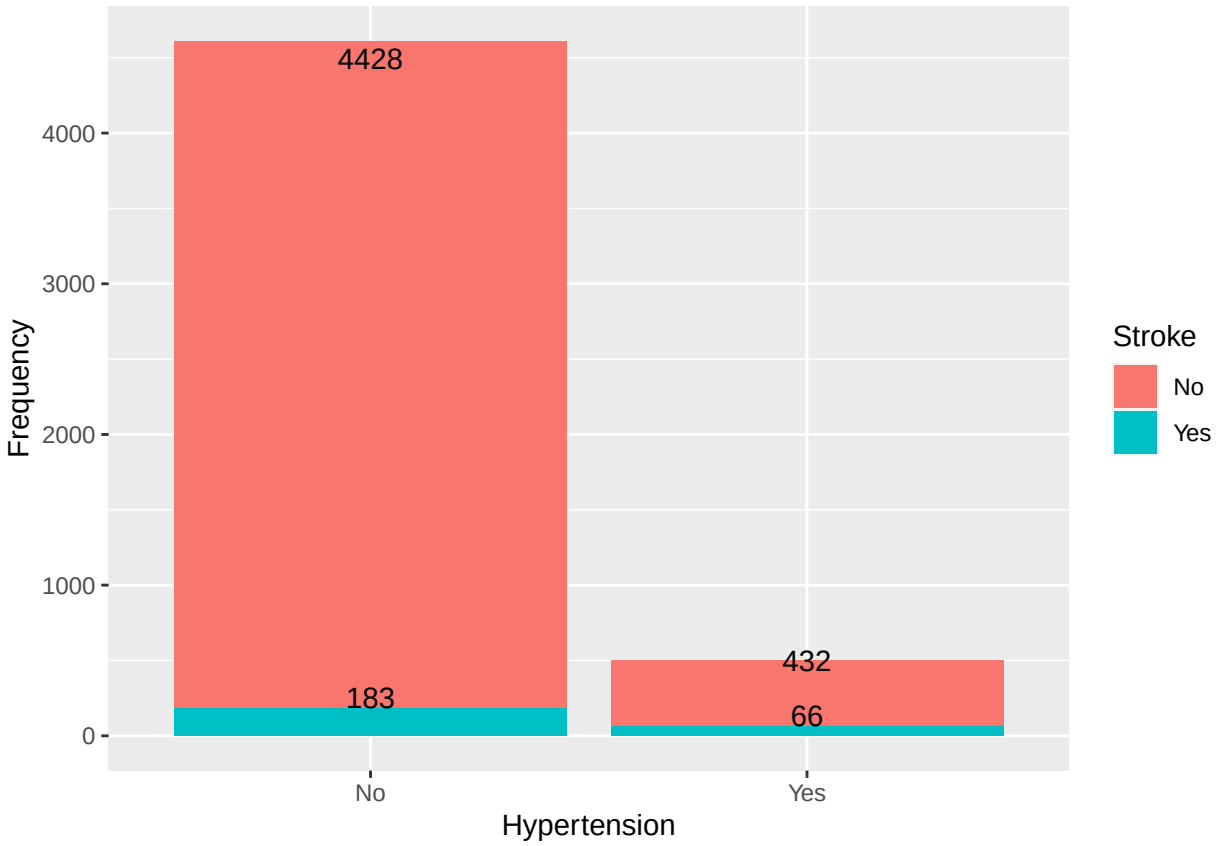


2. Hypertension

```
ht_counts <- as.data.frame(table(stroke_data$hypertension))
ht_counts$Var1 <- if_else(ht_counts$Var1 == 0, 'No','Yes')

ggplot(ht_counts, aes(x = Var1, y = Freq, fill = Var1)) +
        geom_bar(stat = "identity") + theme(legend.position="none") +
        geom_text(aes(label = Freq), vjust = 0) +
        labs(x ="Hypertension", y = "Frequency")
```

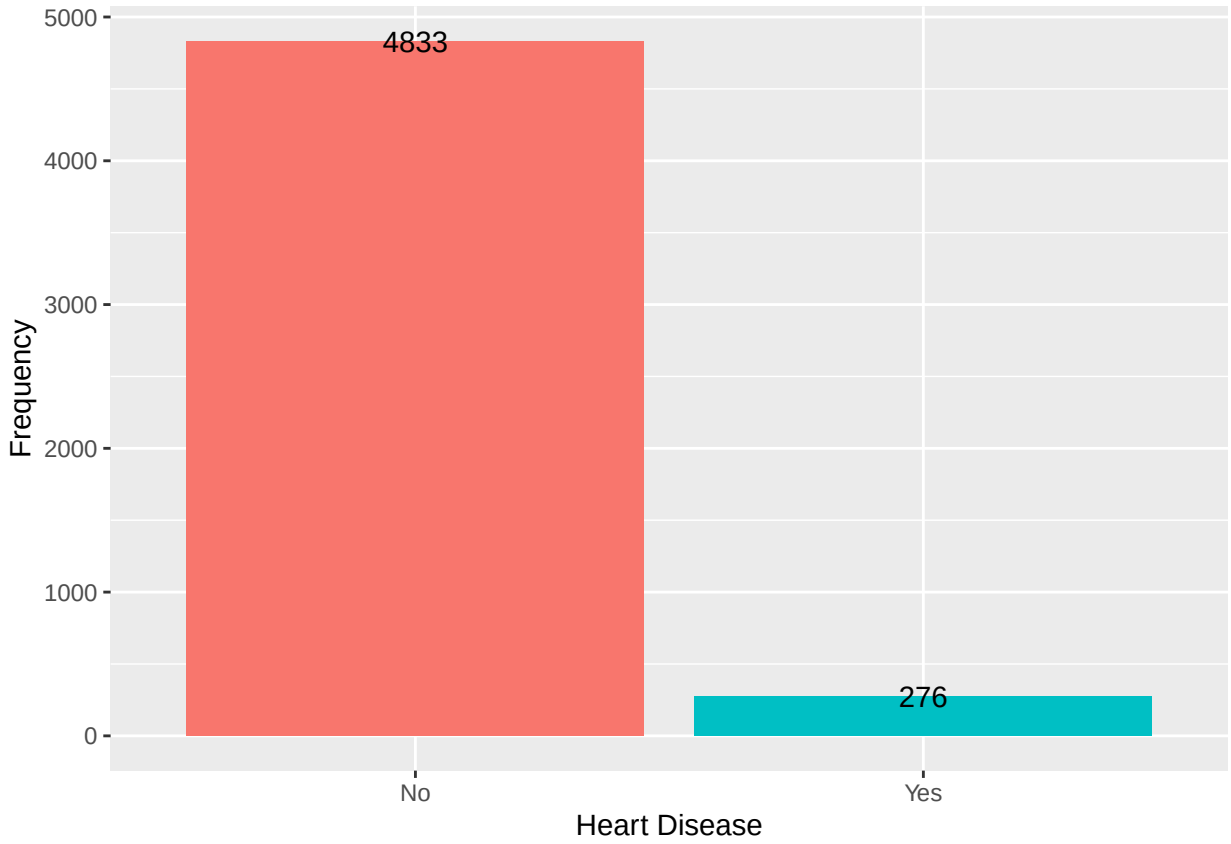



```
hypstr_counts <- as.data.frame(table(stroke_data$hypertension, stroke_data$stroke))
hypstr_counts$Var1 <- if_else(hypstr_counts$Var1 == 0, 'No','Yes')
hypstr_counts$Var2 <- if_else(hypstr_counts$Var2 == 0, 'No','Yes')

hypo_plot <- ggplot(hypstr_counts, aes(x = Var1, y = Freq, fill = Var2)) +
        geom_bar(stat = "identity")  +
        geom_text(aes(label = Freq), vjust = 0) +
        labs(x ="Hypertension", y = "Frequency", fill = 'Stroke')
hypo_plot
```

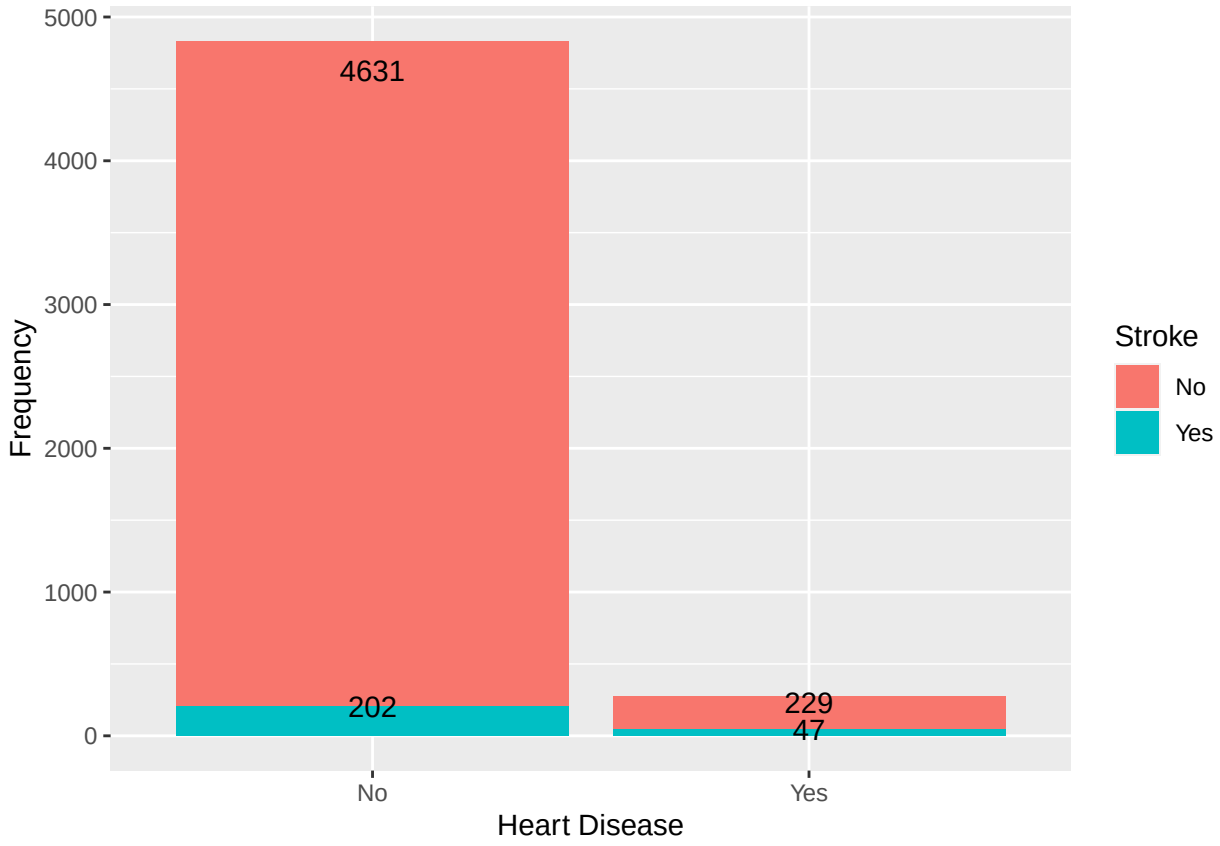


3. Heart Disease

```
hd_counts <- as.data.frame(table(stroke_data$heart_disease))
hd_counts$Var1 <- if_else(hd_counts$Var1 == 0, 'No','Yes')

ggplot(hd_counts, aes(x = Var1, y = Freq, fill = Var1)) +
        geom_bar(stat = "identity") + theme(legend.position="none") +
        geom_text(aes(label = Freq)) +
        labs(x ="Heart Disease", y = "Frequency")
```

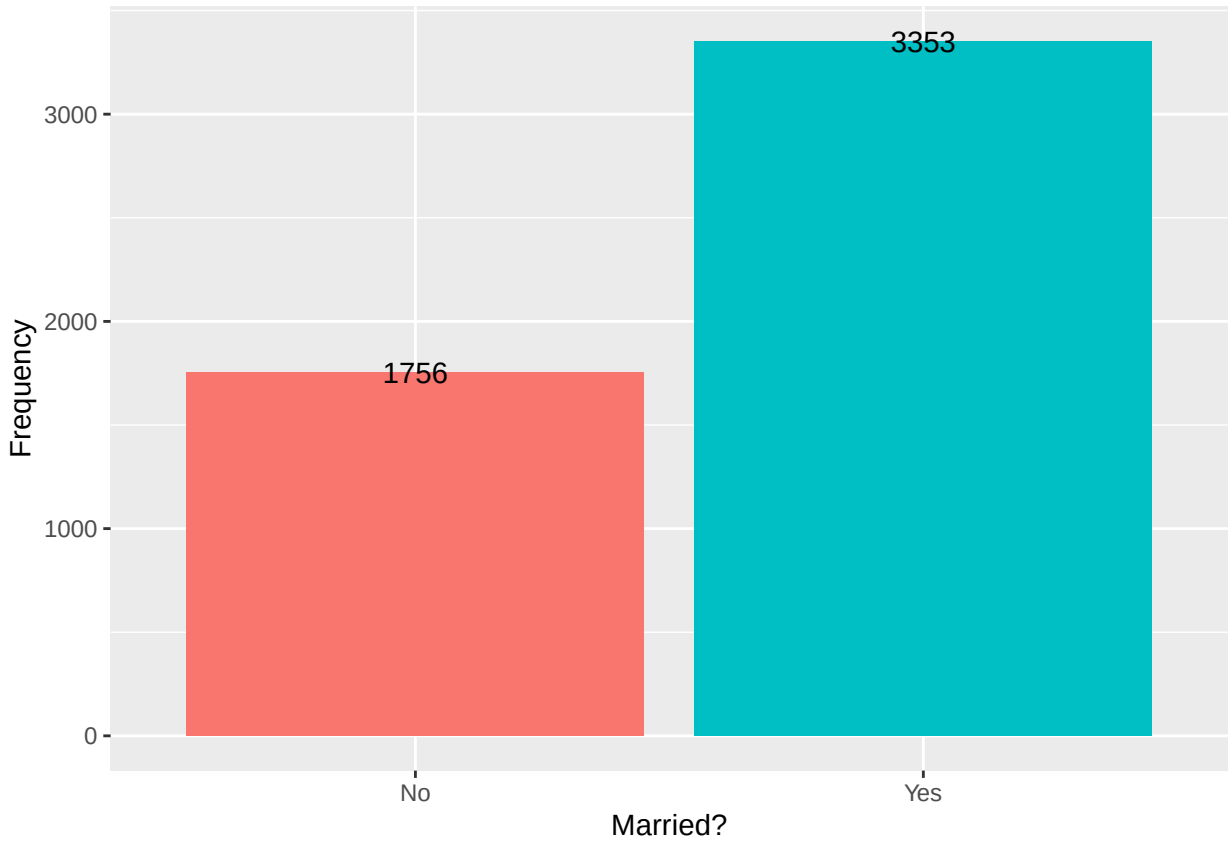



```
hdstr_counts <- as.data.frame(table(stroke_data$heart_disease, stroke_data$stroke))
hdstr_counts$Var1 <- if_else(hdstr_counts$Var1 == 0, 'No','Yes')
hdstr_counts$Var2 <- if_else(hdstr_counts$Var2 == 0, 'No','Yes')

heart_plot <- ggplot(hdstr_counts, aes(x = Var1, y = Freq, fill = Var2)) +
        geom_bar(stat = "identity")  +
        geom_text(aes(label = Freq)) +
        labs(x ="Heart Disease", y = "Frequency", fill = 'Stroke')
heart_plot
```

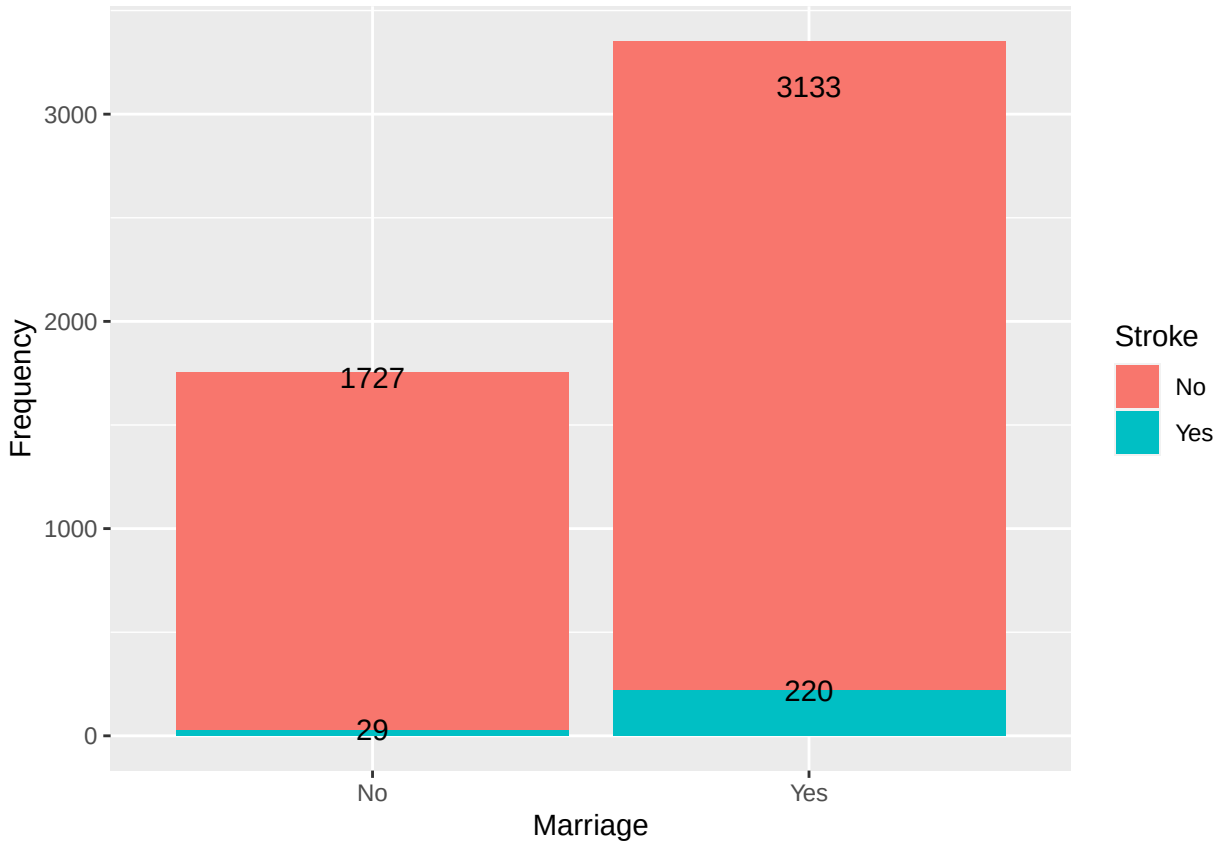


4. Ever Married

```
m_counts <- as.data.frame(table(stroke_data$ever_married))

ggplot(m_counts, aes(x = Var1, y = Freq, fill = Var1)) +
        geom_bar(stat = "identity") + theme(legend.position="none") +
        geom_text(aes(label = Freq)) +
        labs(x ="Married?", y = "Frequency")
```

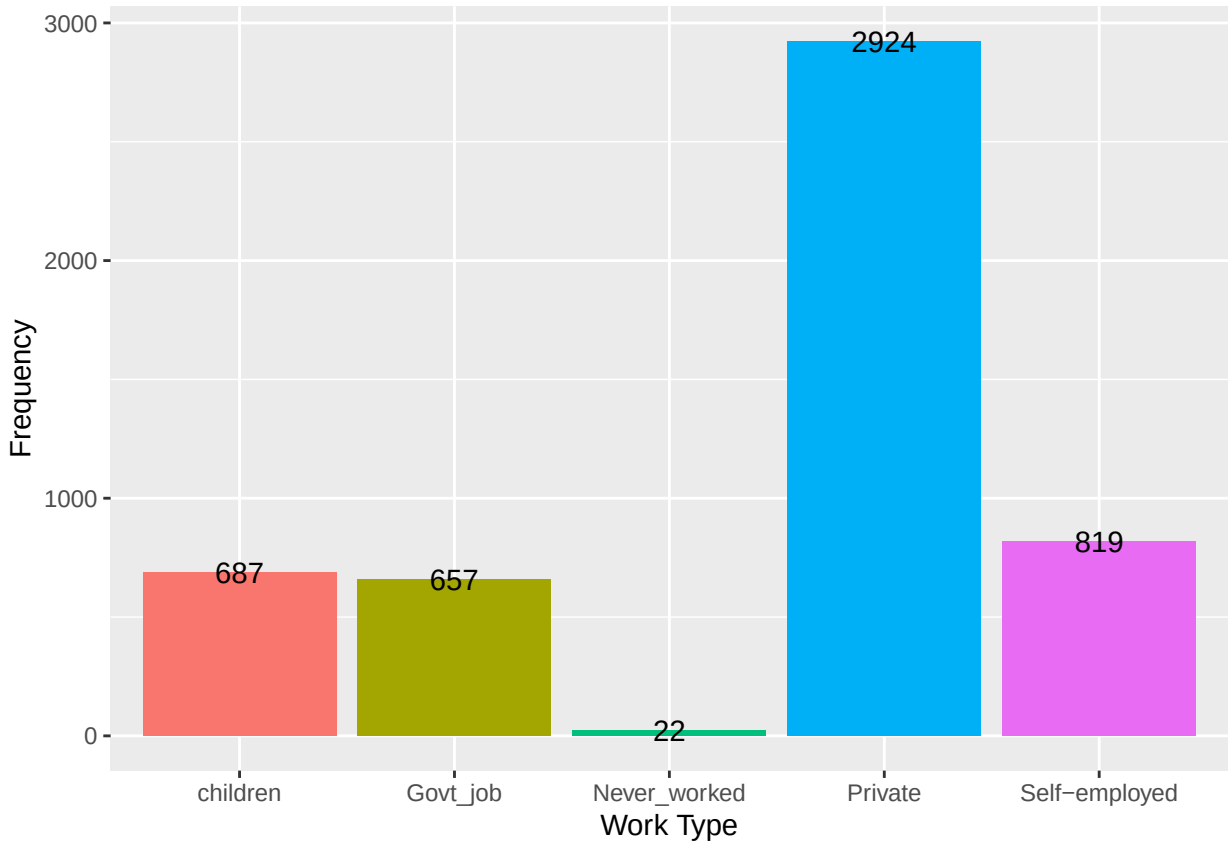



```
mstr_counts <- as.data.frame(table(stroke_data$ever_married, stroke_data$stroke))
mstr_counts$Var2 <- if_else(mstr_counts$Var2 == 0, 'No','Yes')

married_plot <- ggplot(mstr_counts, aes(x = Var1, y = Freq, fill = Var2)) +
        geom_bar(stat = "identity")  +
        geom_text(aes(label = Freq)) +
        labs(x ="Marriage", y = "Frequency", fill = 'Stroke')

married_plot
```

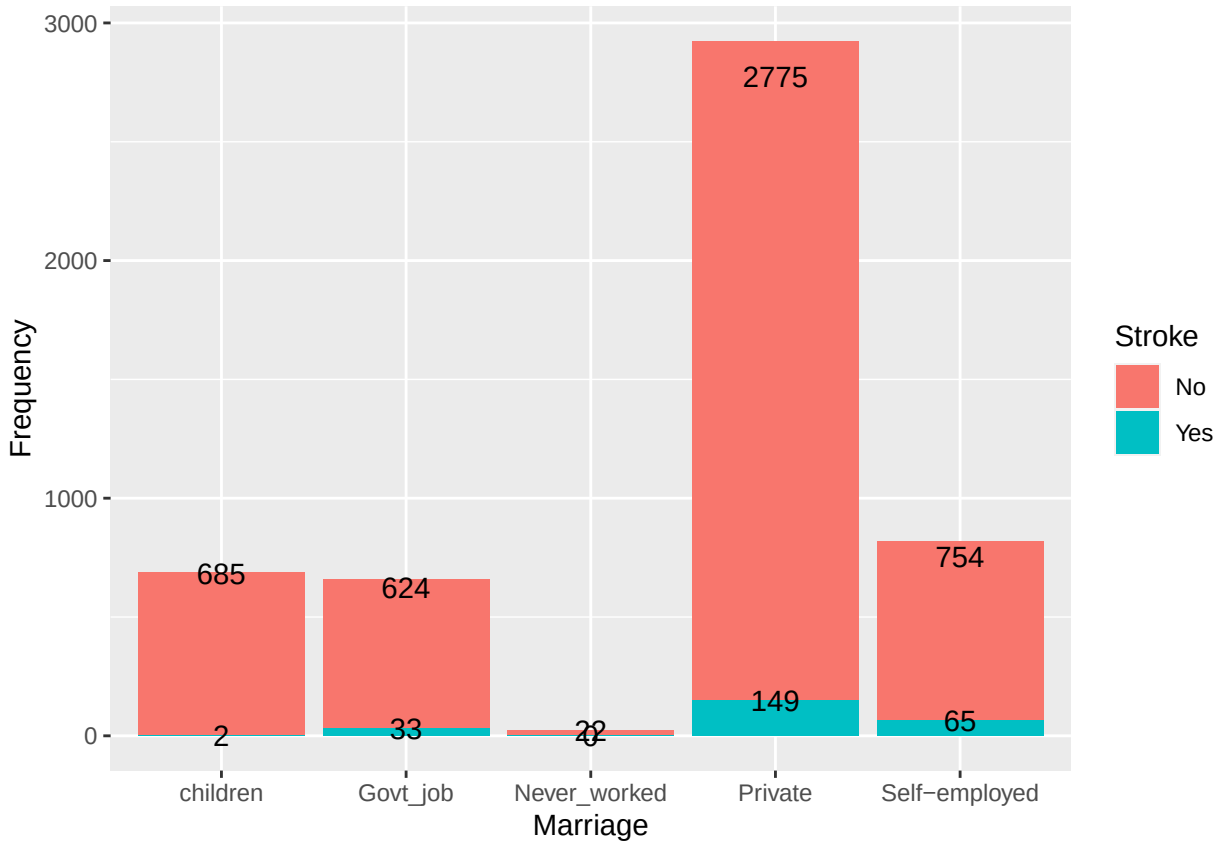


5. Work Type

```
wt_counts <- as.data.frame(table(stroke_data$work_type))

ggplot(wt_counts, aes(x = Var1, y = Freq, fill = Var1)) +
        geom_bar(stat = "identity") + theme(legend.position="none") +
        geom_text(aes(label = Freq)) +
        labs(x ="Work Type", y = "Frequency")
```

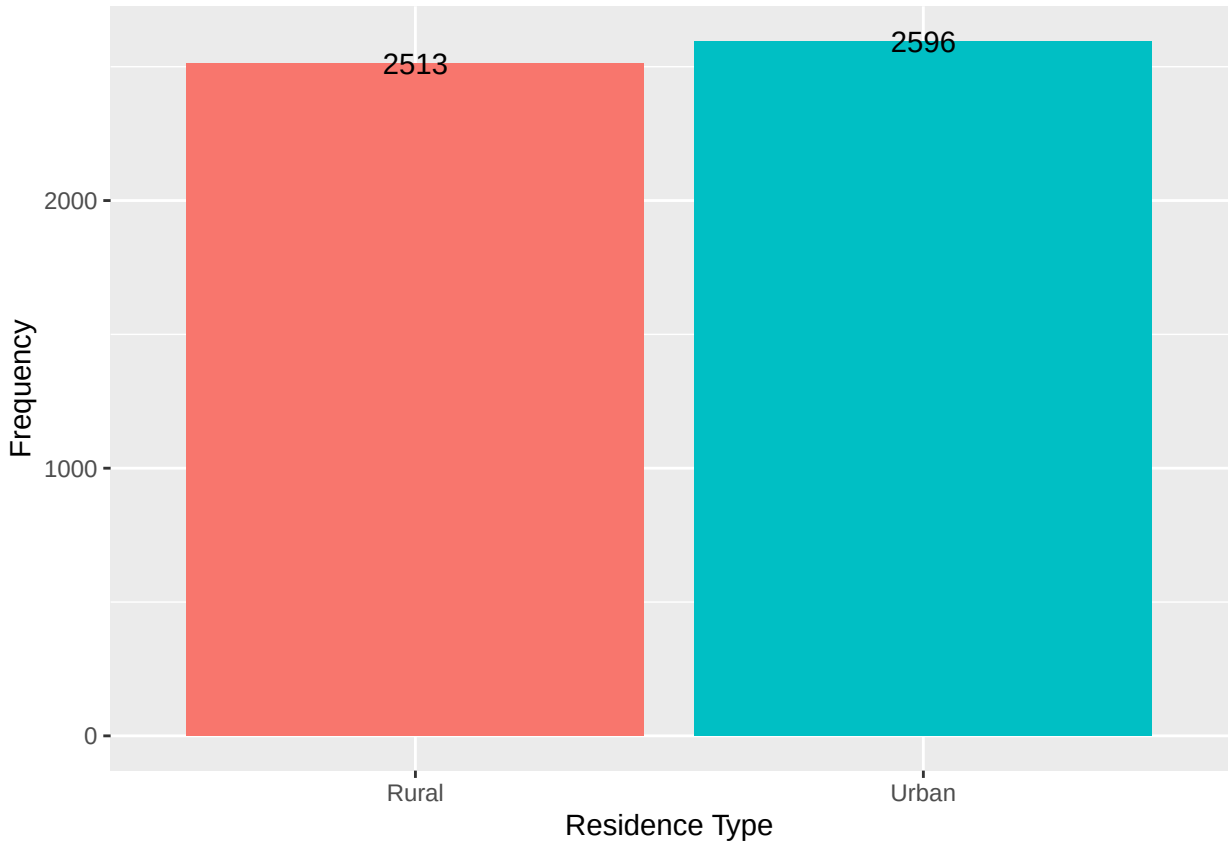



```
wtstr_counts <- as.data.frame(table(stroke_data$work_type, stroke_data$stroke))
wtstr_counts$Var2 <- if_else(wtstr_counts$Var2 == 0, 'No','Yes')

work_plot <- ggplot(wtstr_counts, aes(x = Var1, y = Freq, fill = Var2)) +
        geom_bar(stat = "identity")  +
        geom_text(aes(label = Freq)) +
        labs(x ="Marriage", y = "Frequency", fill = 'Stroke')

work_plot
```

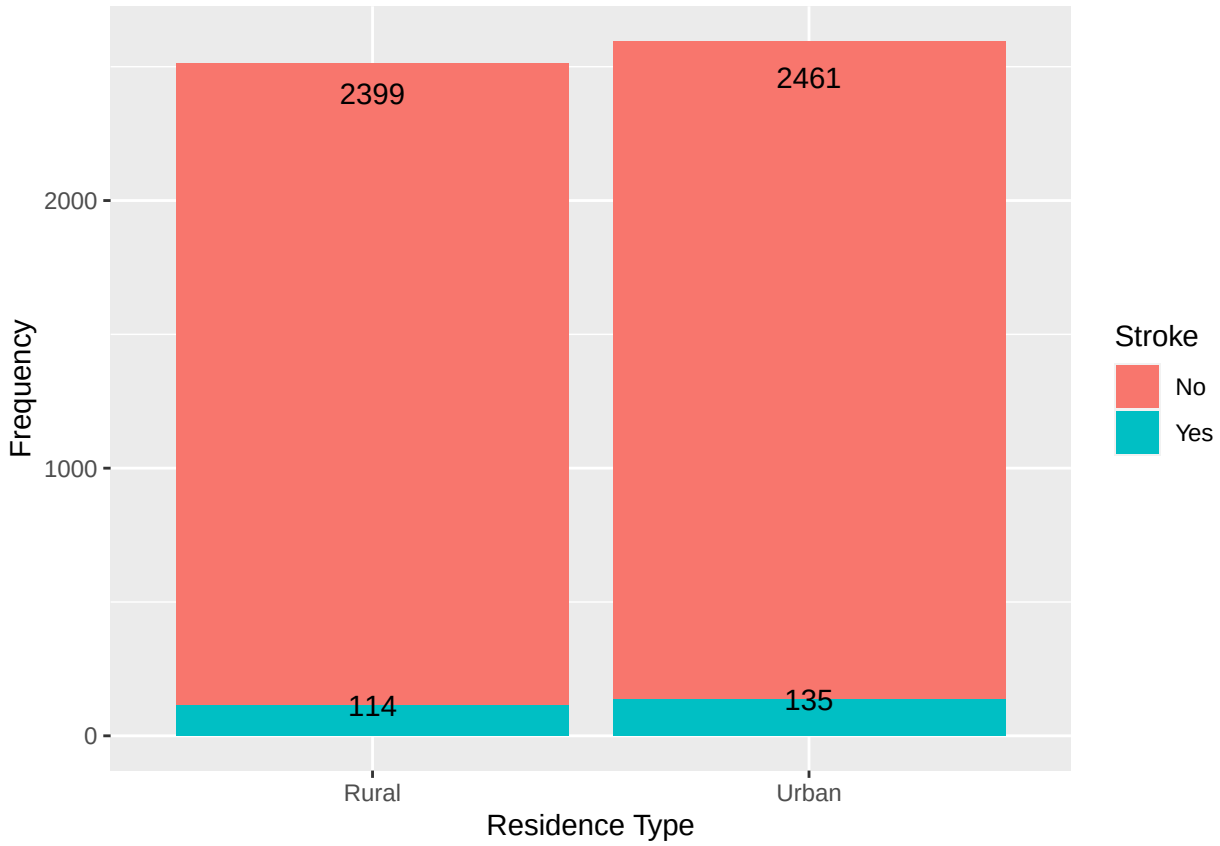


6. Residence Type

```
rt_counts <- as.data.frame(table(stroke_data$Residence_type))

ggplot(rt_counts, aes(x = Var1, y = Freq, fill = Var1)) +
        geom_bar(stat = "identity") + theme(legend.position="none") +
        geom_text(aes(label = Freq)) +
        labs(x ="Residence Type", y = "Frequency")
```

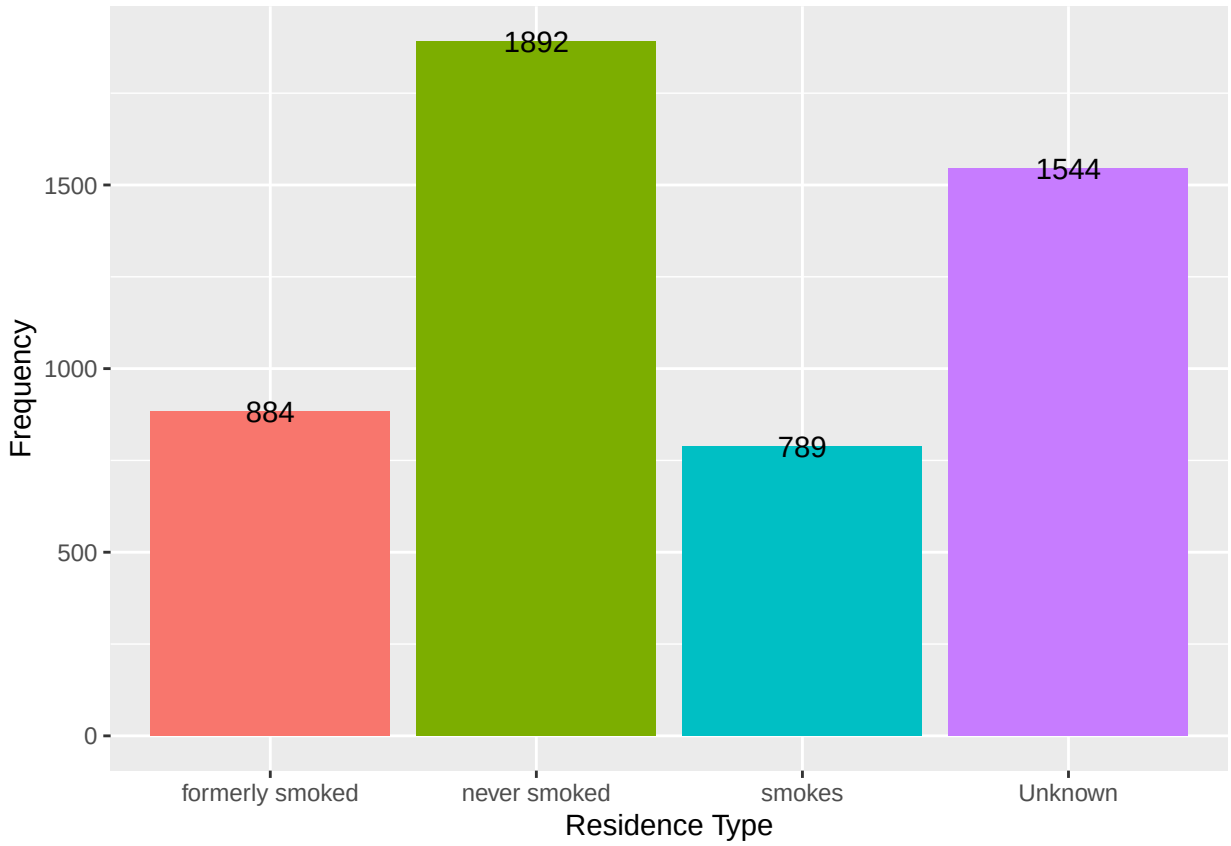



```
rtstr_counts <- as.data.frame(table(stroke_data$Residence_type, stroke_data$stroke))
rtstr_counts$Var2 <- if_else(rtstr_counts$Var2 == 0, 'No','Yes')

residence_plot <- ggplot(rtstr_counts, aes(x = Var1, y = Freq, fill = Var2)) +
        geom_bar(stat = "identity")  +
        geom_text(aes(label = Freq)) +
        labs(x ="Residence Type", y = "Frequency", fill = 'Stroke')
residence_plot
```

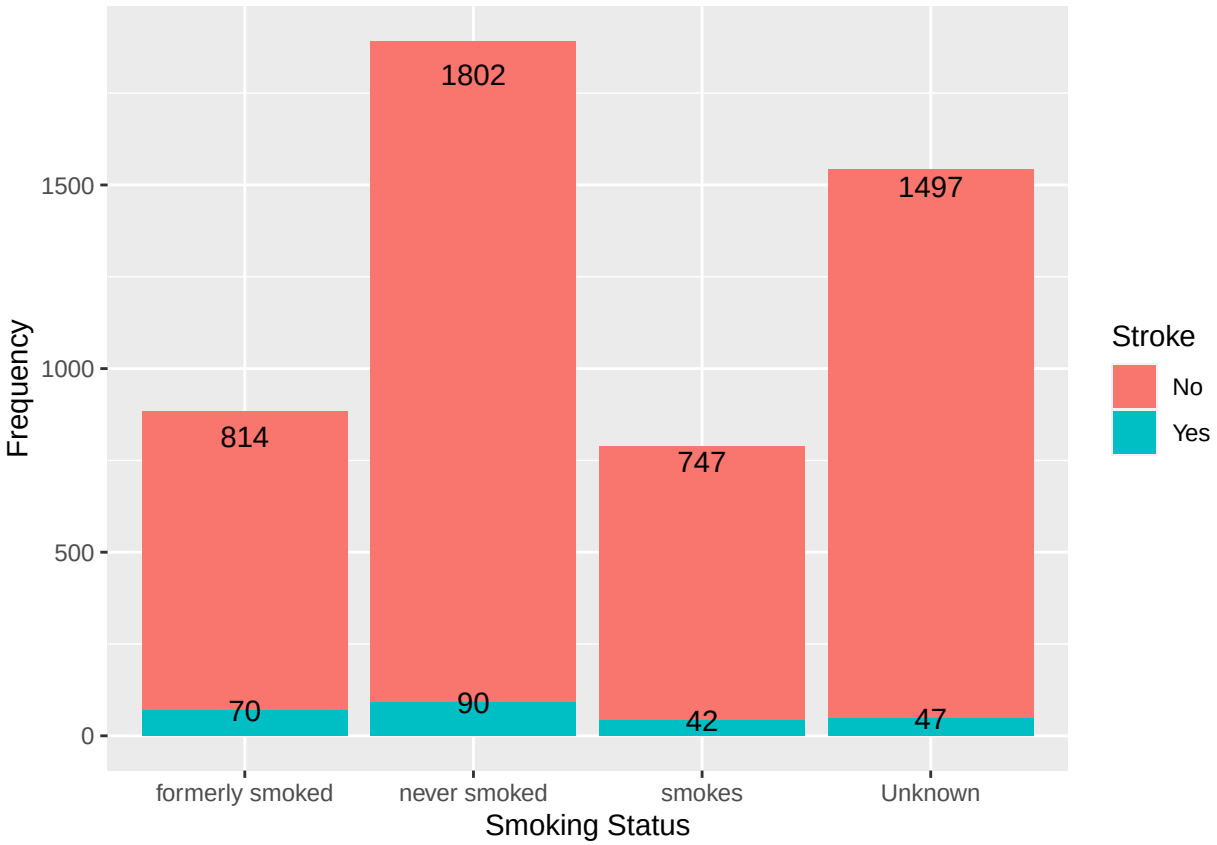


7. Smoking Status

```
ss_counts <- as.data.frame(table(stroke_data$smoking_status))

ggplot(ss_counts, aes(x = Var1, y = Freq, fill = Var1)) +
        geom_bar(stat = "identity") + theme(legend.position="none") +
        geom_text(aes(label = Freq)) +
        labs(x ="Residence Type", y = "Frequency")
```




```
ssstr_counts <- as.data.frame(table(stroke_data$smoking_status, stroke_data$stroke))
ssstr_counts$Var2 <- if_else(ssstr_counts$Var2 == 0, 'No','Yes')

smoke_plot <- ggplot(ssstr_counts, aes(x = Var1, y = Freq, fill = Var2)) +
        geom_bar(stat = "identity")  +
        geom_text(aes(label = Freq)) +
        labs(x ="Smoking Status", y = "Frequency", fill = 'Stroke')
smoke_plot
```

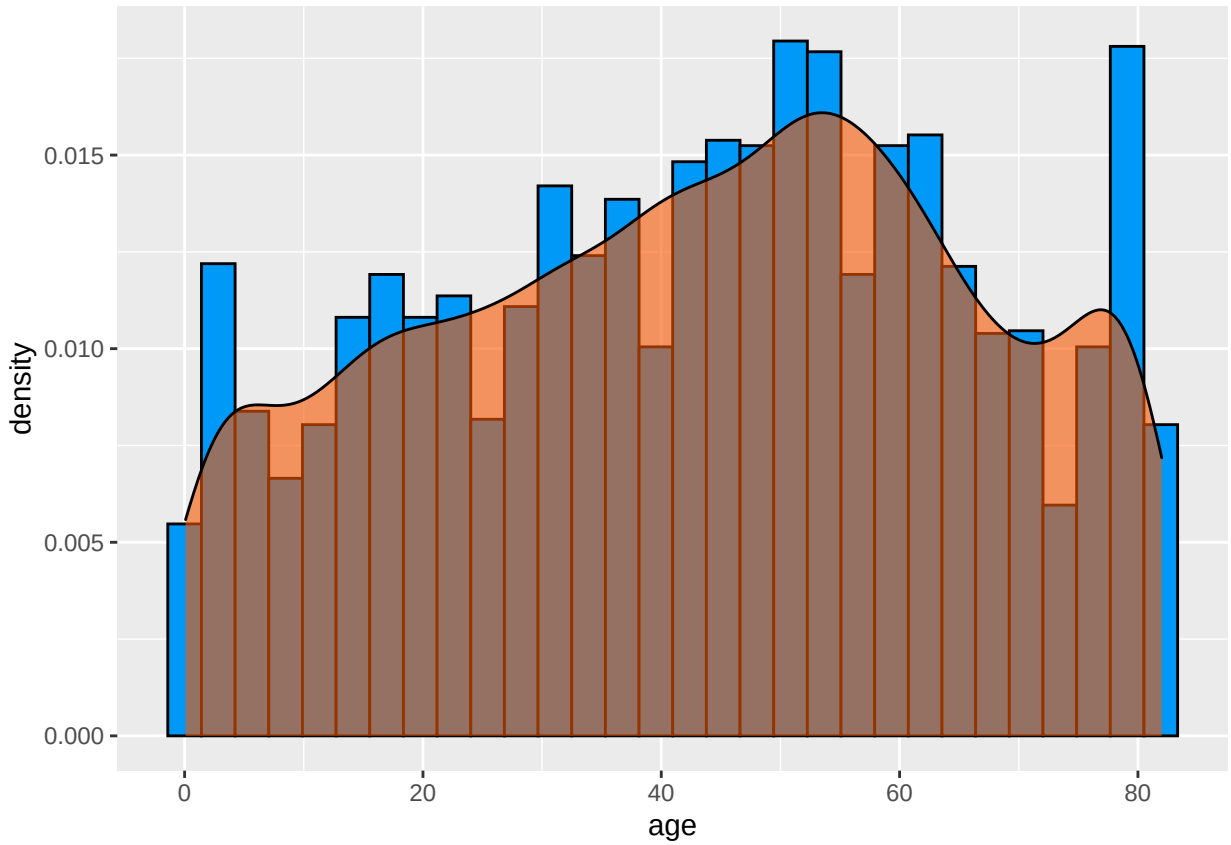


Continous Variables

8. Age

```
#reference: https://www.r-bloggers.com/2021/11/how-to-make-stunning-histograms-in-r-a-complete-guide-wi
age_plot <- ggplot(stroke_data, aes(x=age)) +
  geom_histogram(aes(y = ..density..),color = "#000000", fill = "#0099F8") +
  geom_density(color = "#000000", fill = "#F85700", alpha = 0.6)
age_plot
```

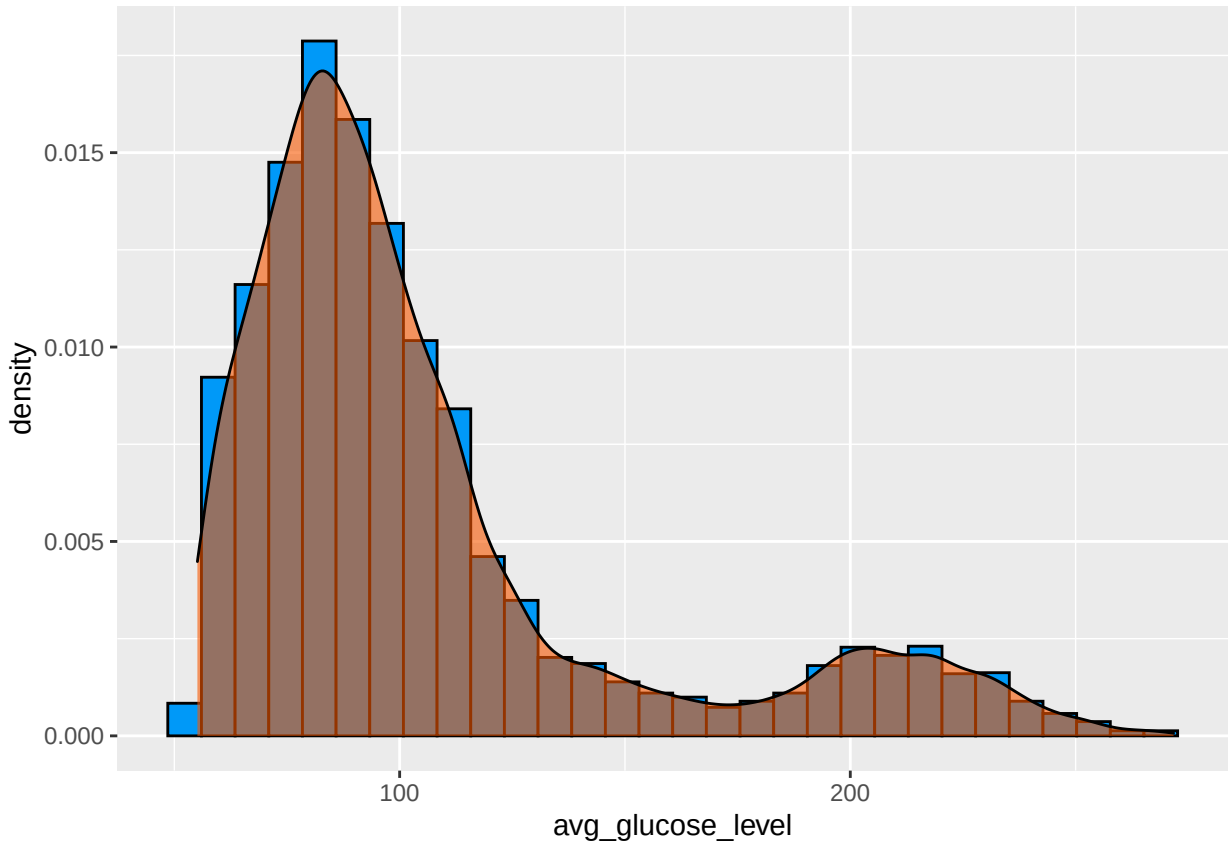
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



9. Avg Glucose Level

```
glucose_plot <- ggplot(stroke_data, aes(x=avg_glucose_level)) +
  geom_histogram(aes(y = ..density..),color = "#000000", fill = "#0099F8") +
  geom_density(color = "#000000", fill = "#F85700", alpha = 0.6)

glucose_plot
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



10. BMI

```
#reference: https://www.r-bloggers.com/2021/11/how-to-make-stunning-histograms-in-r-a-complete-guide-wi
stroke_data$bmi <- as.integer(stroke_data$bmi)
```

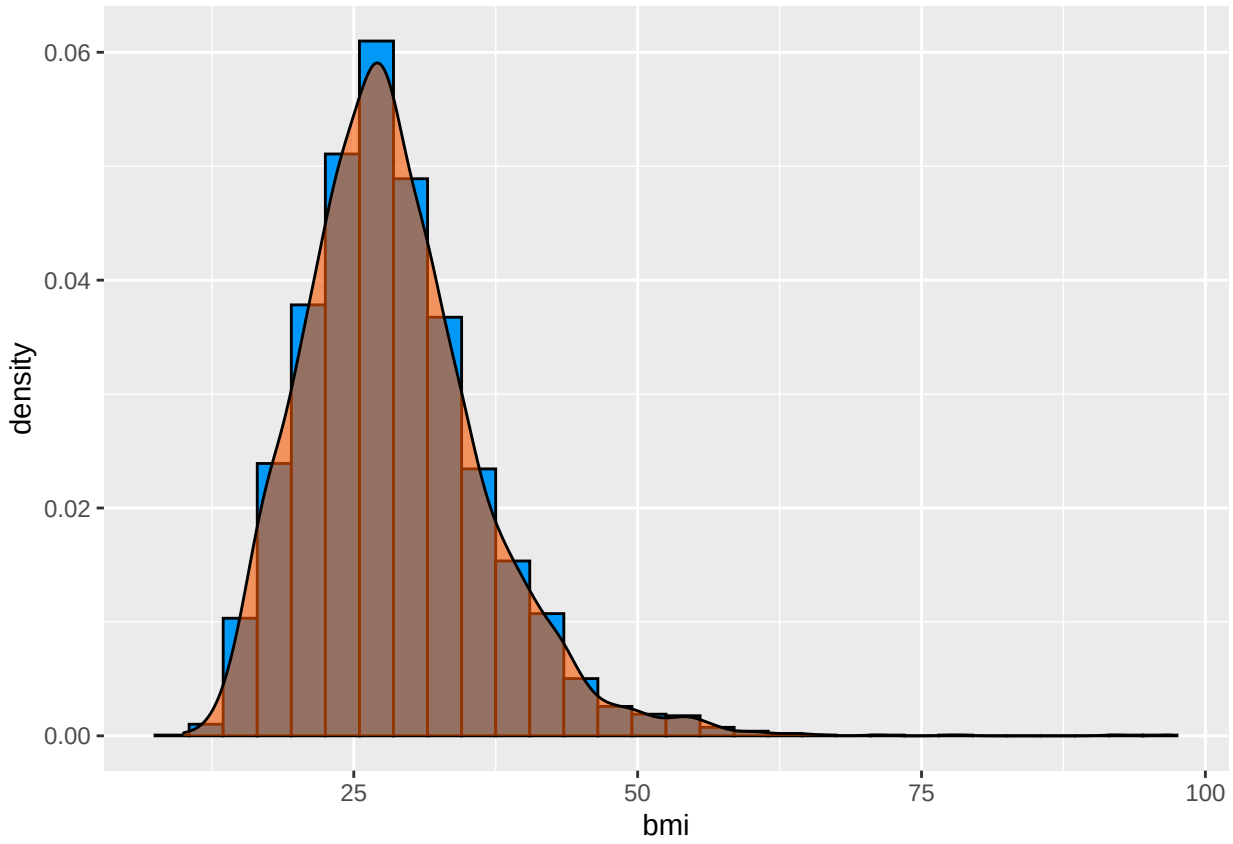
```
## Warning: NAs introduced by coercion
```

```
bmi_plot <- ggplot(stroke_data, aes(x=bmi)) +
  geom_histogram(aes(y = ..density..),color = "#000000", fill = "#0099F8") +
  geom_density(color = "#000000", fill = "#F85700", alpha = 0.6)
bmi_plot
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 201 rows containing non-finite values (stat_bin).
```

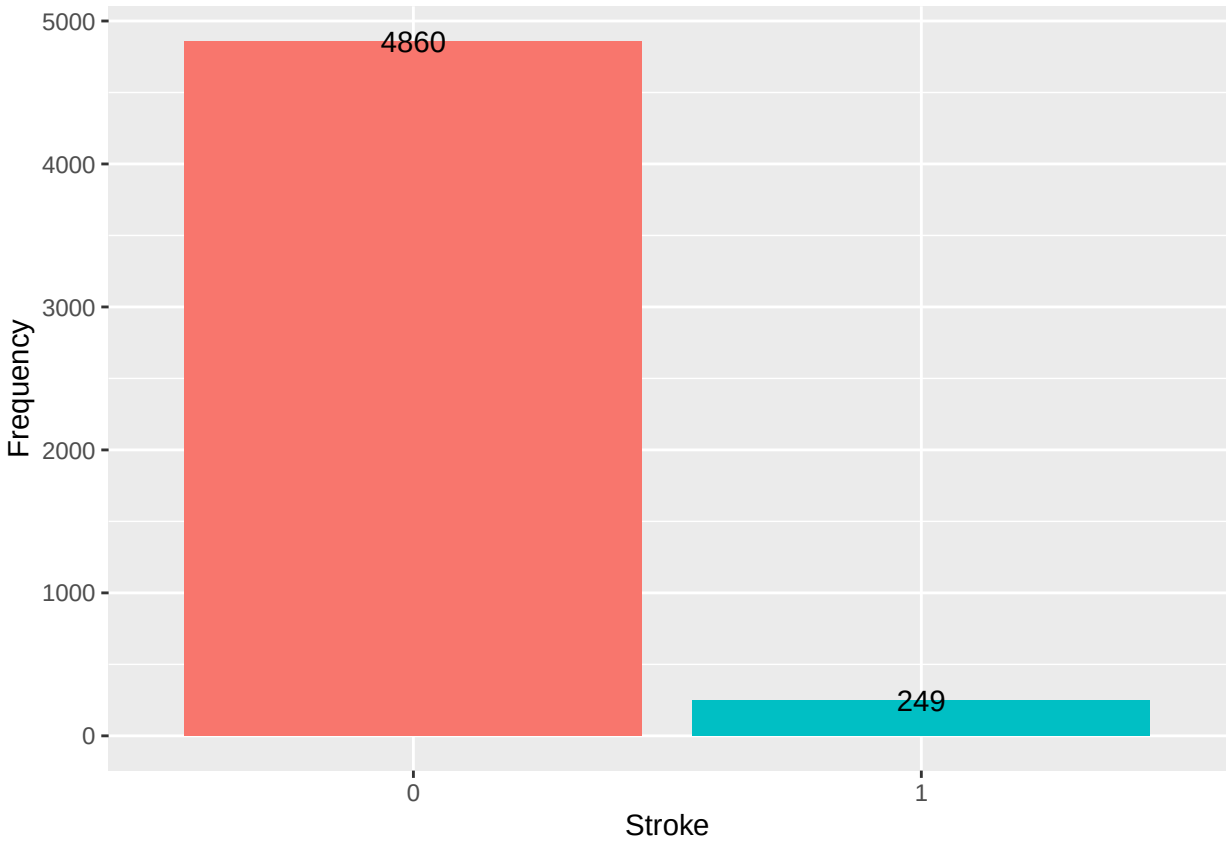
```
## Warning: Removed 201 rows containing non-finite values (stat_density).
```



Dependent Vairable

11. Stroke

```
s_counts <- as.data.frame(table(stroke_data$stroke))

stroke_plot <- ggplot(s_counts, aes(x = Var1, y = Freq, fill = Var1)) +
        geom_bar(stat = "identity") + theme(legend.position="none") +
        geom_text(aes(label = Freq)) +
        labs(x ="Stroke", y = "Frequency")
stroke_plot
```



As we can see from the above graph, Our dependent variable stroke is highly imbalanced, Which is true in a realistic scenario as for a sample population, Number of people suffering from stroke is less.

Here, we have consolidated distributions of all categorical variables with respect to whether the patient has stroke or not

```
require(gridExtra)
```

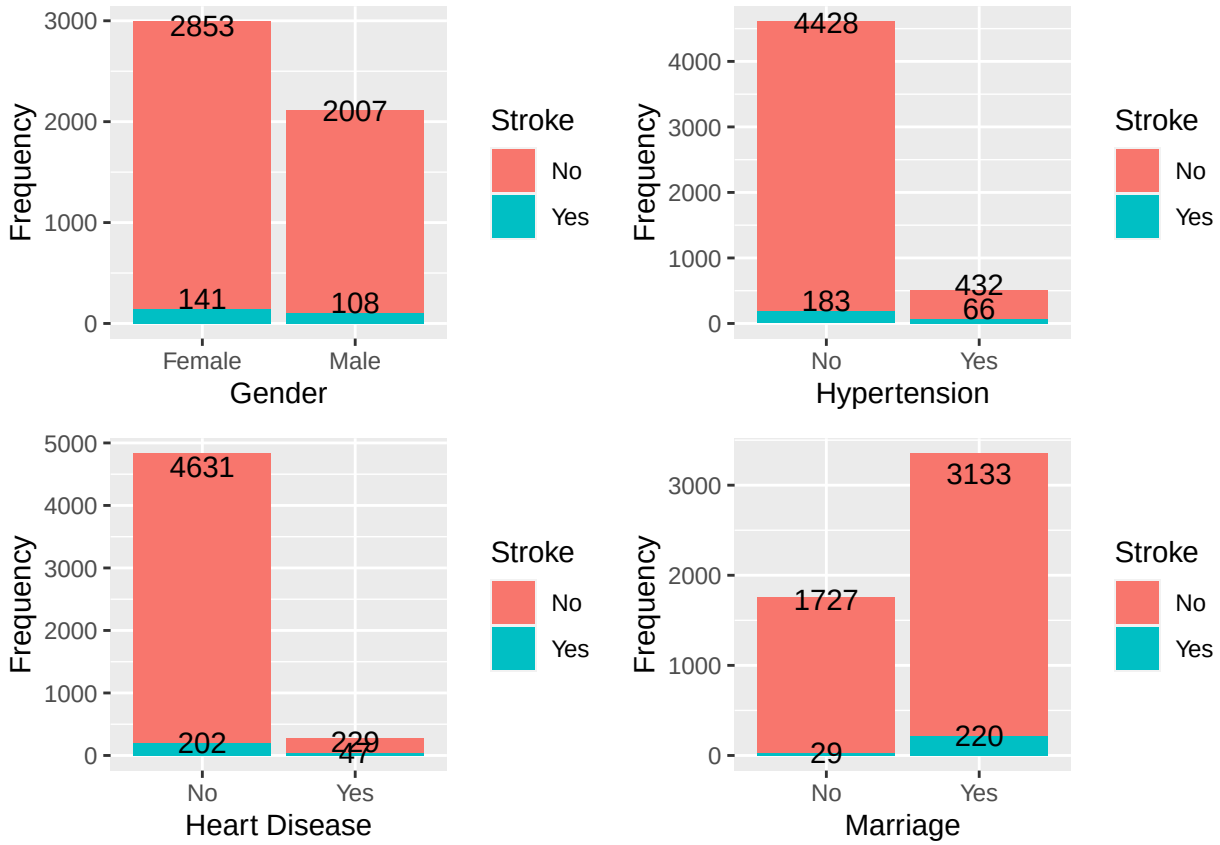
```
## Loading required package: gridExtra
```

```
##
## Attaching package: 'gridExtra'
```

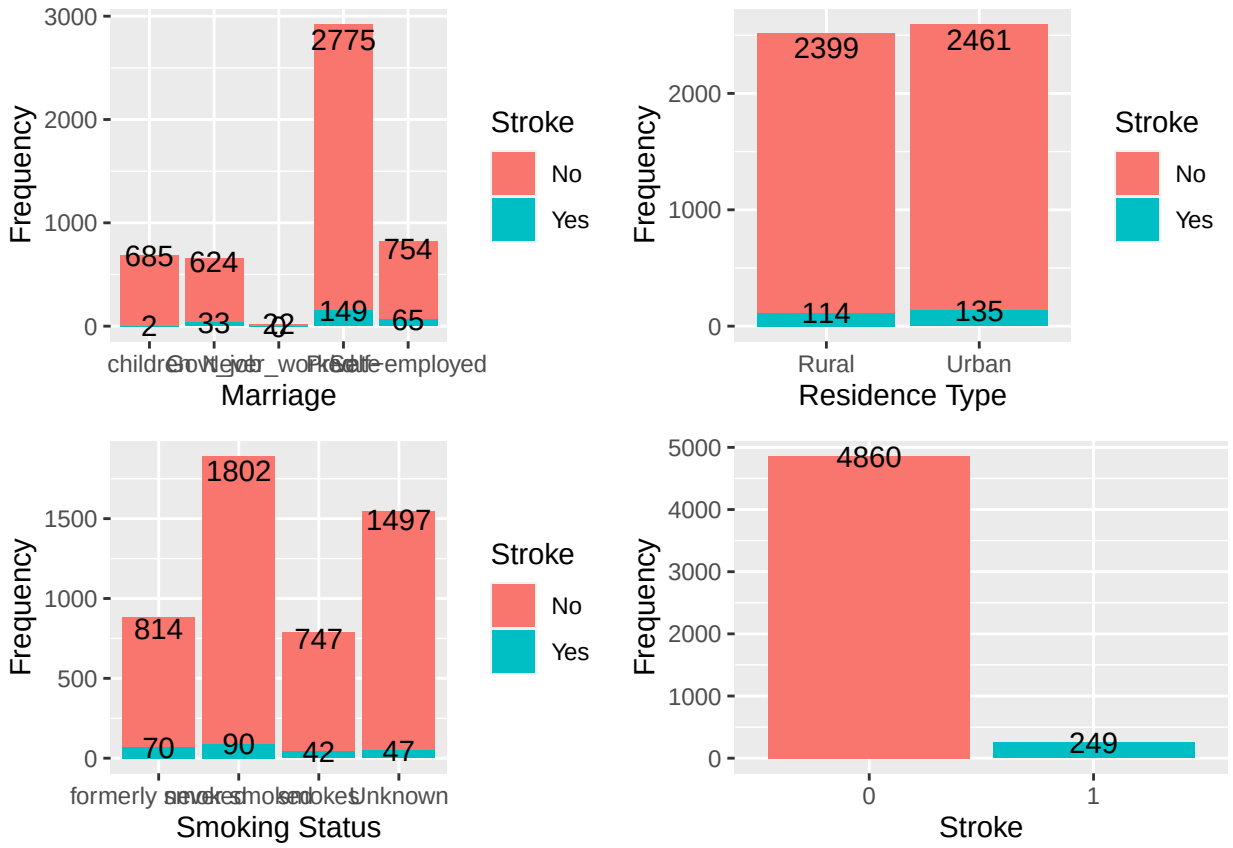
```
## The following object is masked from 'package:randomForest':
##
##     combine
```

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

```
grid.arrange(gender_plot,hypo_plot,heart_plot,married_plot, nrow = 2, ncol=2)
```




```
grid.arrange(work_plot,residence_plot,smoke_plot,stroke_plot, nrow = 2, ncol=2)
```



Factorizing all the categorical variables

```
# reference for one hot coding - https://datatricks.co.uk/one-hot-encoding-in-r-three-simple-methods

stroke_data$gender = factor(stroke_data$gender,levels = c('Male', 'Female'),labels = c(0,1))
stroke_data$ever_married = factor(stroke_data$ever_married,levels = c('No', 'Yes'),labels = c(0,1))
stroke_data$Residence_type = factor(stroke_data$Residence_type,levels = c('Rural', 'Urban'),labels = c(

stroke_data$hypertension = factor(stroke_data$hypertension,levels = c('0', '1'),labels = c(0,1))

stroke_data$heart_disease = factor(stroke_data$heart_disease,levels = c('0', '1'),labels = c(0,1))

stroke_data$smoking_status = factor(stroke_data$smoking_status,levels = c("formerly smoked", "never smol
stroke_data$work_type = factor(stroke_data$work_type,levels = c("children", "Govt_job", "Never_worked",

stroke_data['bmi'] <- as.numeric(stroke_data$bmi)
stroke_data$stroke<-as.factor(stroke_data$stroke)
```

Check for NA's in the Dataset

```
#Columnwise percentage of rows which are NA
colMeans(is.na(stroke_data))*100
```

```
##               id           gender              age     hypertension
##         0.000000         0.000000         0.000000         0.000000
##    heart_disease     ever_married        work_type   Residence_type
##         0.000000         0.000000         0.000000         0.000000
```

```
## avg_glucose_level               bmi    smoking_status            stroke
##          0.000000          3.934234          0.000000          0.000000
```

Analysis for imputation of missing values

Since there very few missing values and patient data is very sensitive to loose we will go ahead imputing the missing values with a selected paramter.

```
mean_bmi <- mean(na.omit(stroke_data$bmi))
median_bmi <- median(na.omit(stroke_data$bmi))

# estimating the mode value of the bmi column
# reference - https://www.tutorialspoint.com/r/r_mean_median_mode.htm

getmode <- function(v) {
   uniqv <- unique(v)
   index <-which.max(tabulate(match(v, uniqv))) #index of the most occuring value
   uniqv[index]
}

mode_bmi <- getmode(na.omit(stroke_data$bmi))


print(mean_bmi)
```

```
## [1] 28.44988
```

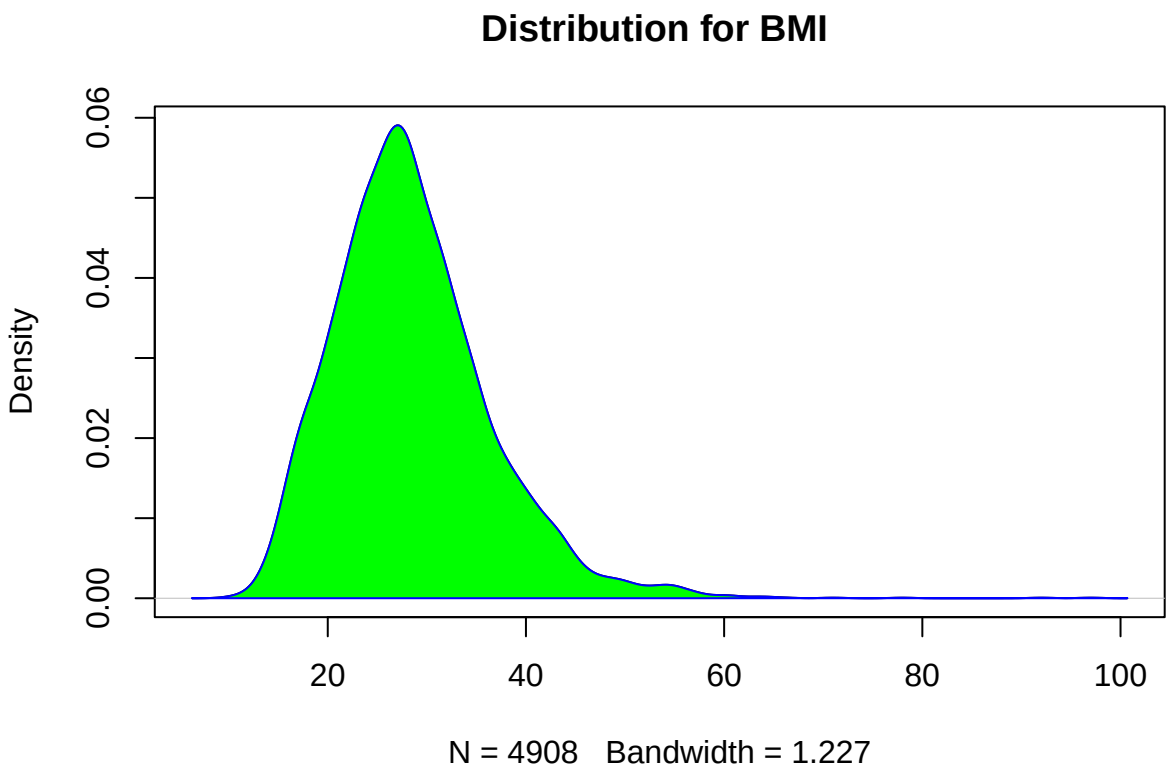
```
print(median_bmi)
```

```
## [1] 28
```

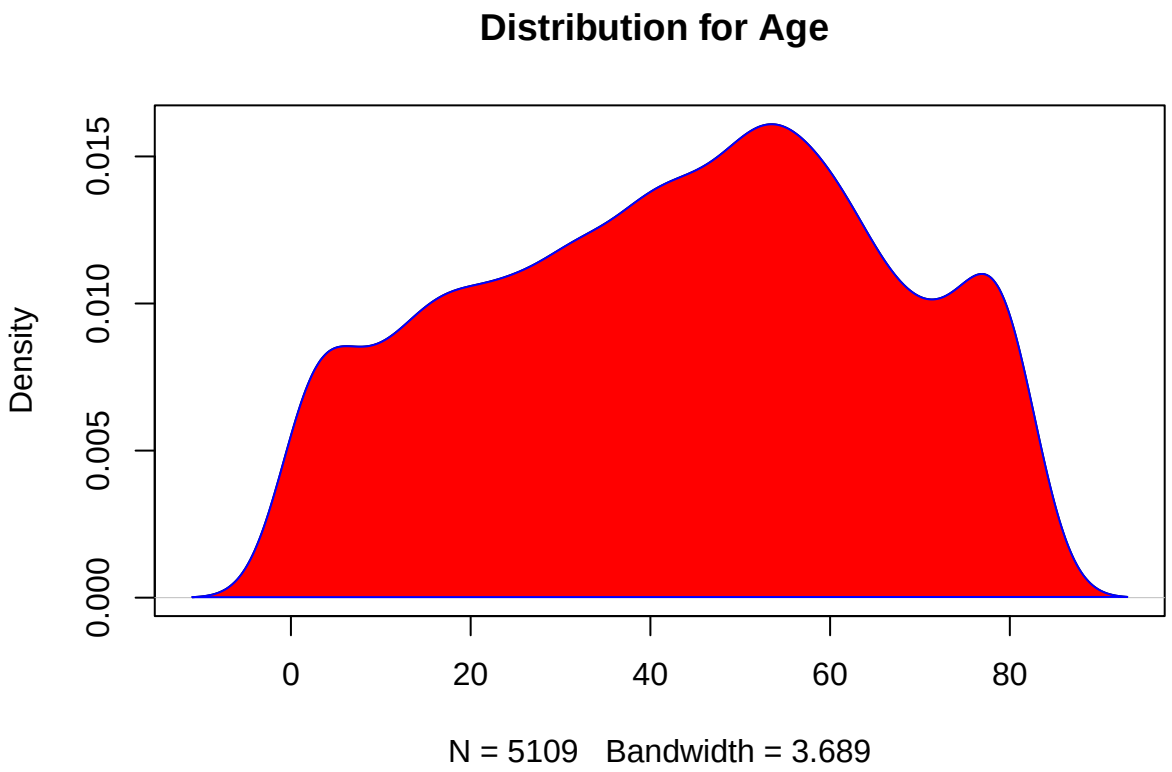
```
print(mode_bmi)
```

```
## [1] 28
```

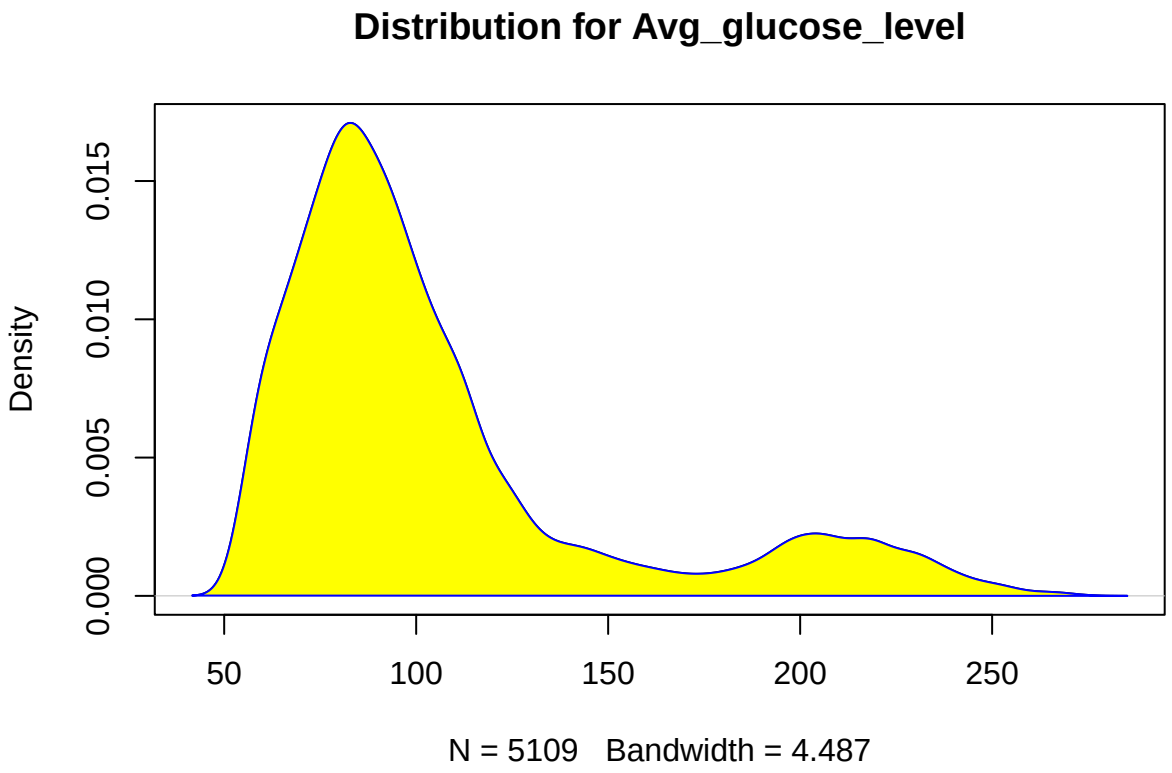
```
# distribution for BMI
d <- density(na.omit(stroke_data$bmi))
plot(d, main="Distribution for BMI ")
polygon(d, col="green", border="blue")
```




```
# distribution for age
d <- density(na.omit(stroke_data$age))
plot(d, main="Distribution for Age ")
polygon(d, col="red", border="blue")
```



```
# distribution for avg_glucose_level
d <- density(na.omit(stroke_data$avg_glucose_level))
plot(d, main="Distribution for Avg_glucose_level")
polygon(d, col="yellow", border="blue")
```



Imputation of missing values with mean of the values from the column 'bmi'

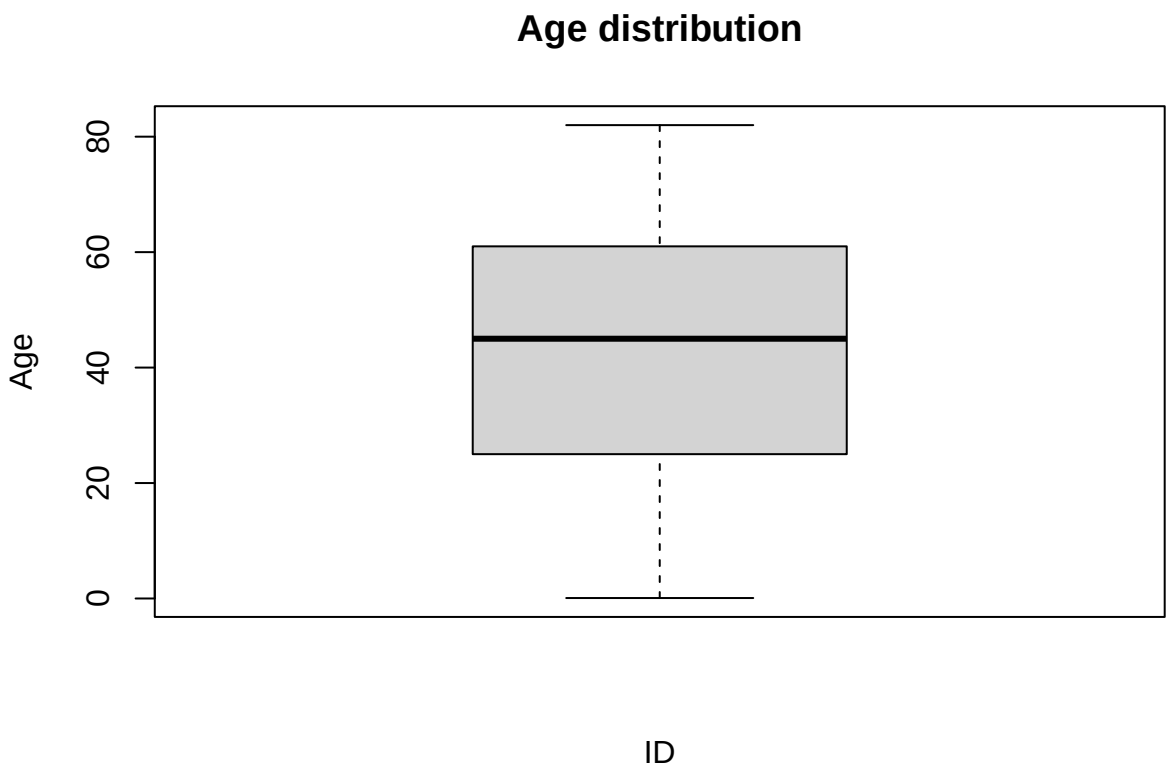
Since the distribution is uniform and the mean, median and mode values are almost the same, we will go ahead with mean imputation for the missing values in the 'bmi' variable column

```
stroke_data$bmi[is.na(stroke_data$bmi)] <- mean(stroke_data$bmi, na.rm = TRUE)
stroke_data <- na.omit(stroke_data)
str(stroke_data)
```

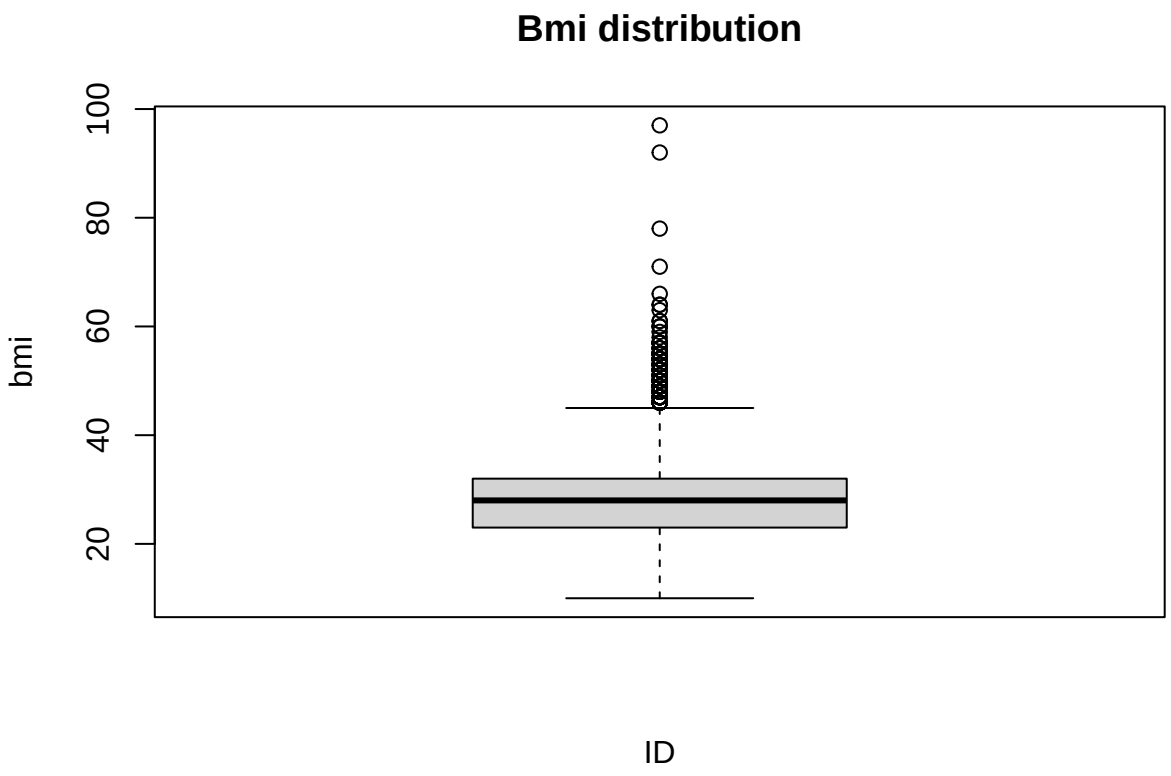
```
## 'data.frame':    5109 obs. of  12 variables:
##  $ id               : int  9046 51676 31112 60182 1665 56669 53882 10434 27419 60491 ...
##  $ gender           : Factor w/ 2 levels "0","1": 1 2 1 2 2 1 1 2 2 2 ...
##  $ age              : num  67 61 80 49 79 81 74 69 59 78 ...
##  $ hypertension     : Factor w/ 2 levels "0","1": 1 1 1 1 2 1 2 1 1 1 ...
##  $ heart_disease    : Factor w/ 2 levels "0","1": 2 1 2 1 1 1 2 1 1 1 ...
##  $ ever_married     : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 1 2 2 ...
##  $ work_type        : Factor w/ 5 levels "0","1","2","3",..: 4 5 4 4 5 4 4 4 4 4 ...
##  $ Residence_type   : Factor w/ 2 levels "0","1": 2 1 1 2 1 2 1 2 1 2 ...
##  $ avg_glucose_level: num  229 202 106 171 174 ...
##  $ bmi              : num  36 28.4 32 34 24 ...
##  $ smoking_status   : Factor w/ 4 levels "0","1","2","3": 1 2 2 3 2 1 2 2 4 4 ...
##  $ stroke           : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
```

Checking for outliers

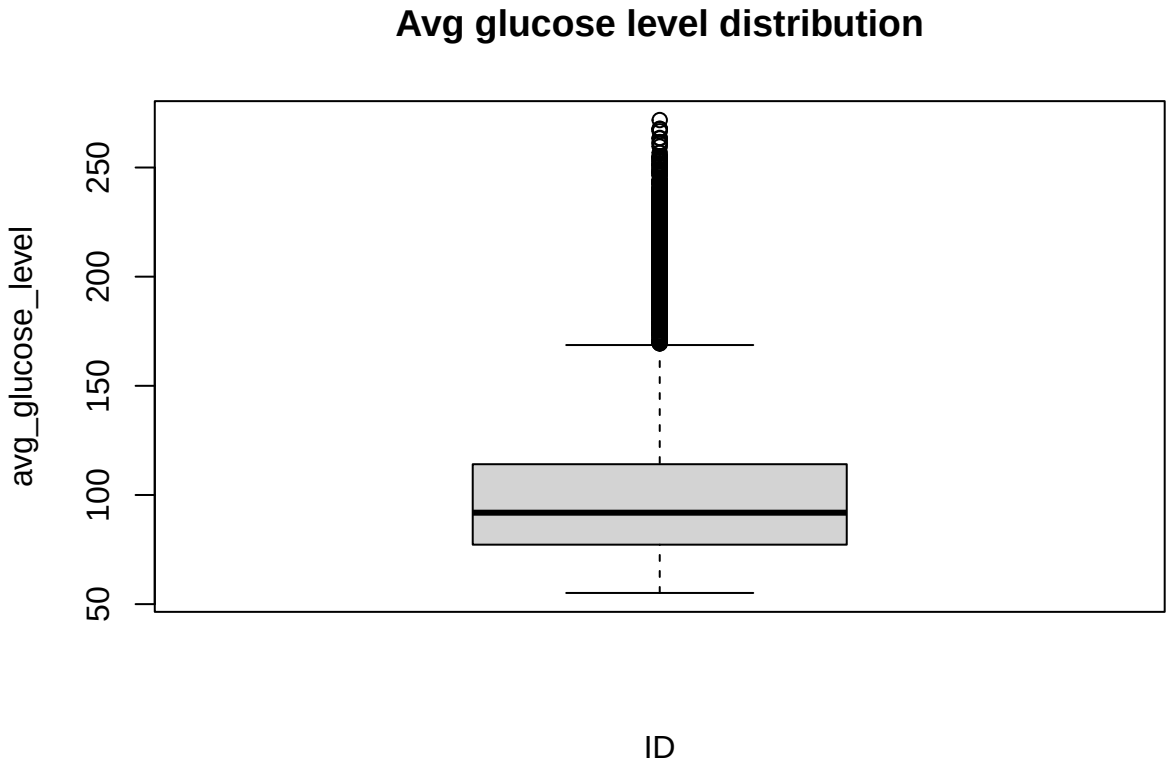
```
#boxplot before outlier removal
boxplot(stroke_data$age, main="Age distribution",
   xlab="ID", ylab="Age")
```




```
boxplot(stroke_data$bmi, main="Bmi distribution",
   xlab="ID", ylab="bmi")
```




```
boxplot(stroke_data$avg_glucose_level, main="Avg glucose level distribution",
   xlab="ID", ylab="avg_glucose_level")
```



Outlier removal based on IQR

All the values beyond 1.5 times the Inter-quartile range (IQR) will be considered as outliers and removed.

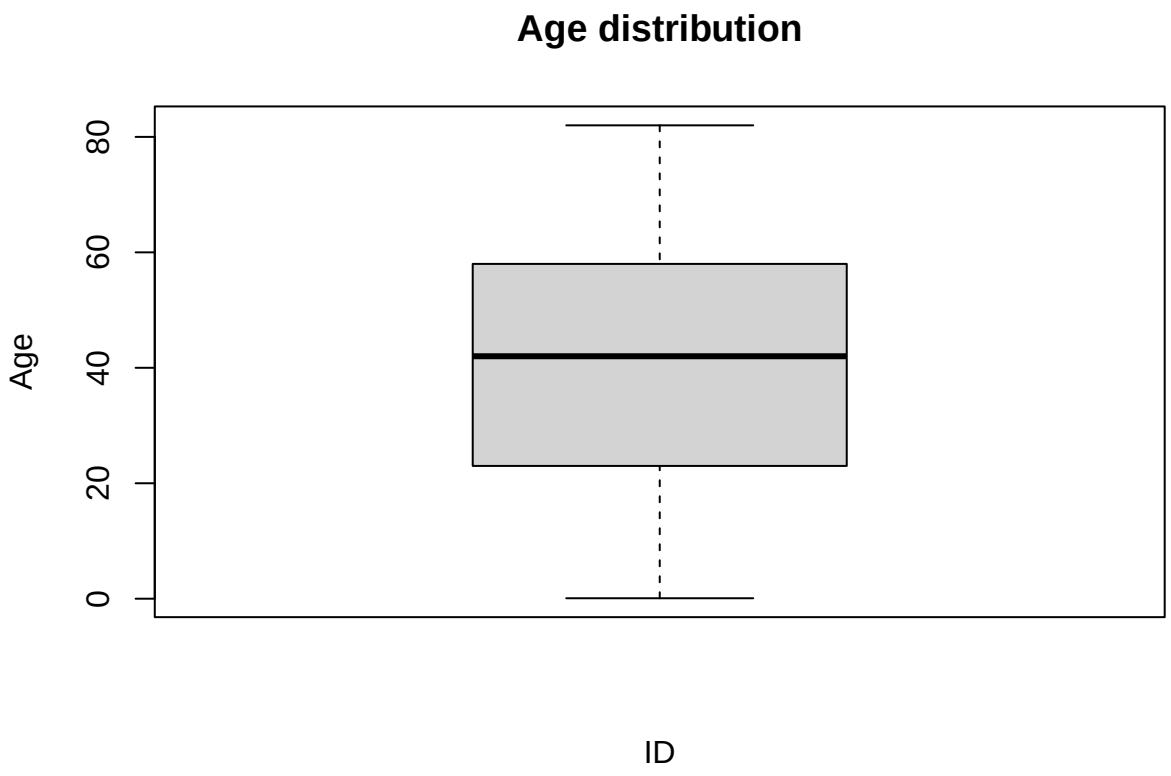
```r
#IQR

Q_age <- quantile(stroke_data$age, probs=c(.25, .75), na.rm = FALSE)
iqr_age <- IQR(stroke_data$age)

Q_bmi<- quantile(stroke_data$bmi, probs=c(.25, .75), na.rm = FALSE)
iqr_bmi <- IQR(stroke_data$bmi)

Q_avg_glucose_level<- quantile(stroke_data$avg_glucose_level, probs=c(.25, .75), na.rm = FALSE)
iqr_avg_glucose_level <- IQR(stroke_data$avg_glucose_level)


stroke_data_clean<- subset(stroke_data,
                    stroke_data$age > (Q_age[1] - 1.5*iqr_age) &
                    stroke_data$age < (Q_age[2] + 1.5*iqr_age) &
                    stroke_data$bmi > (Q_bmi[1] - 1.5*iqr_bmi) &
                    stroke_data$bmi < (Q_bmi[2] + 1.5*iqr_bmi) &
                    stroke_data$avg_glucose_level > (Q_avg_glucose_level[1] - 1.5*iqr_avg_glucose_level
                    stroke_data$avg_glucose_level < (Q_avg_glucose_level[2]+1.5*iqr_avg_glucose_level))

str(stroke_data_clean)
```

```
## 'data.frame':    4385 obs. of  12 variables:
##  $ id               : int  31112 53882 10434 27419 60491 12109 12095 12175 58202 27458 ...
##  $ gender           : Factor w/ 2 levels "0","1": 1 1 2 2 2 2 2 2 2 2 ...
##  $ age              : num  80 74 69 59 78 81 61 54 50 60 ...
##  $ hypertension     : Factor w/ 2 levels "0","1": 1 2 1 1 1 2 1 1 2 1 ...
##  $ heart_disease    : Factor w/ 2 levels "0","1": 2 2 1 1 1 1 2 1 1 1 ...
##  $ ever_married     : Factor w/ 2 levels "0","1": 2 2 1 2 2 2 2 2 2 1 ...
##  $ work_type        : Factor w/ 5 levels "0","1","2","3",..: 4 4 4 4 4 4 2 4 5 4 ...
##  $ Residence_type   : Factor w/ 2 levels "0","1": 1 1 2 1 2 1 1 2 1 2 ...
##  $ avg_glucose_level: num  105.9 70.1 94.4 76.2 58.6 ...
##  $ bmi              : num  32 27 22 28.4 24 ...
##  $ smoking_status   : Factor w/ 4 levels "0","1","2","3": 2 2 2 4 4 2 3 3 2 2 ...
##  $ stroke           : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
```

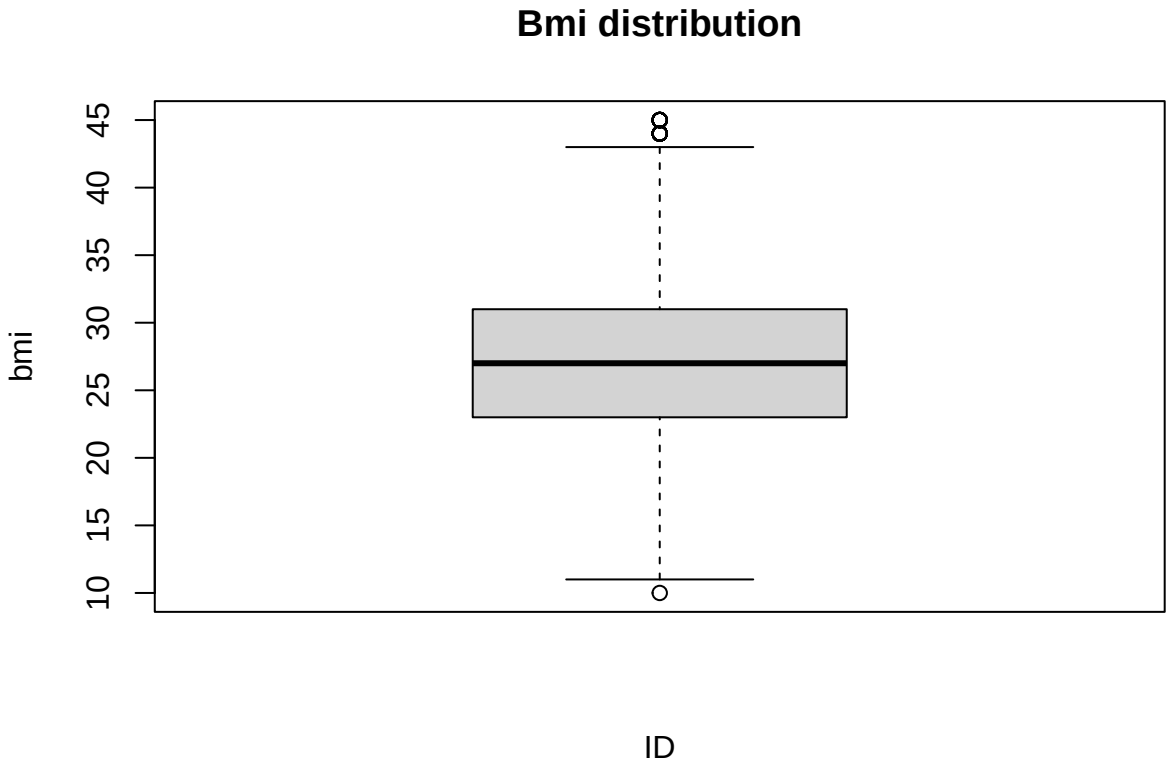
```r
str(stroke_data)
```

```
## 'data.frame':    5109 obs. of  12 variables:
##  $ id               : int  9046 51676 31112 60182 1665 56669 53882 10434 27419 60491 ...
##  $ gender           : Factor w/ 2 levels "0","1": 1 2 1 2 2 1 1 2 2 2 ...
##  $ age              : num  67 61 80 49 79 81 74 69 59 78 ...
##  $ hypertension     : Factor w/ 2 levels "0","1": 1 1 1 1 2 1 2 1 1 1 ...
##  $ heart_disease    : Factor w/ 2 levels "0","1": 2 1 2 1 1 1 2 1 1 1 ...
##  $ ever_married     : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 1 2 2 ...
##  $ work_type        : Factor w/ 5 levels "0","1","2","3",..: 4 5 4 4 5 4 4 4 4 4 ...
##  $ Residence_type   : Factor w/ 2 levels "0","1": 2 1 1 2 1 2 1 2 1 2 ...
##  $ avg_glucose_level: num  229 202 106 171 174 ...
##  $ bmi              : num  36 28.4 32 34 24 ...
##  $ smoking_status   : Factor w/ 4 levels "0","1","2","3": 1 2 2 3 2 1 2 2 4 4 ...
##  $ stroke           : Factor w/ 2 levels "0","1": 2 2 2 2 2 2 2 2 2 2 ...
```

```r
#boxplot after outlier removal
boxplot(stroke_data_clean$age, main="Age distribution",
   xlab="ID", ylab="Age")
```




```
boxplot(stroke_data_clean$bmi, main="Bmi distribution",
   xlab="ID", ylab="bmi")
```




```
boxplot(stroke_data_clean$avg_glucose_level, main="Avg glucose level distribution",
   xlab="ID", ylab="avg_glucose_level")
```

4. Modeling

splitting the data set into train and test set

```
# train test split
train <- sample(1:nrow(stroke_data_clean),nrow(stroke_data_clean)*0.7)
train_data <- stroke_data_clean[train, ]
test_data <- stroke_data_clean[-train, ]
```

Random Forest on unbalanced data set

```
rf <- randomForest(stroke ~., data = train_data, mtry = sqrt(ncol(train_data) - 1), ntree = 500)

# predicting the income value
yhat_rf <- predict(rf, test_data[,-12])

# accuracy
acc_rf = mean(yhat_rf == test_data$stroke)

# classification metrics
cm_rf <- confusionMatrix(yhat_rf, test_data$stroke, mode = "everything", positive="1")
cm_rf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1267   48
##          1    1    0
```

```
##
##                Accuracy : 0.9628
##                  95% CI : (0.9511, 0.9723)
##     No Information Rate : 0.9635
##     P-Value [Acc > NIR] : 0.5957
##
##                   Kappa : -0.0015
##
##  Mcnemar's Test P-Value : 4.983e-11
##
##             Sensitivity : 0.0000000
##             Specificity : 0.9992114
##          Pos Pred Value : 0.0000000
##          Neg Pred Value : 0.9634981
##               Precision : 0.0000000
##                  Recall : 0.0000000
##                      F1 :       NaN
##              Prevalence : 0.0364742
##          Detection Rate : 0.0000000
##    Detection Prevalence : 0.0007599
##       Balanced Accuracy : 0.4996057
##
##        'Positive' Class : 1
##
```

Due to a high class imbalance all the entries are classified as belonging to class 0. This will lead to Precision and Recall value of "0", thus making the F1-score undefined.

Oversampling to handle the imbalance in the data

1. Using ovun.sample oversampling technique

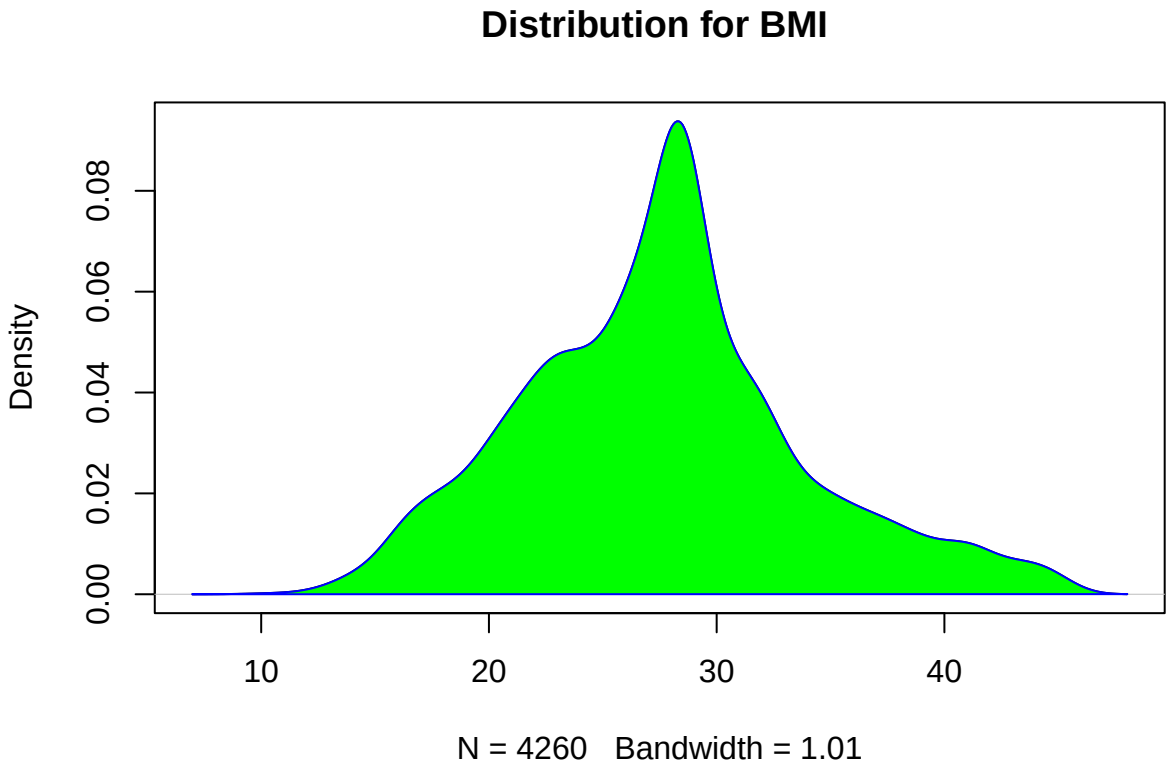
```
data_balanced <- ovun.sample(stroke~ ., data = train_data, p=0.3, method = "over")$data
table(data_balanced$stroke)
```

```
##
##    0    1
## 2952 1308
```

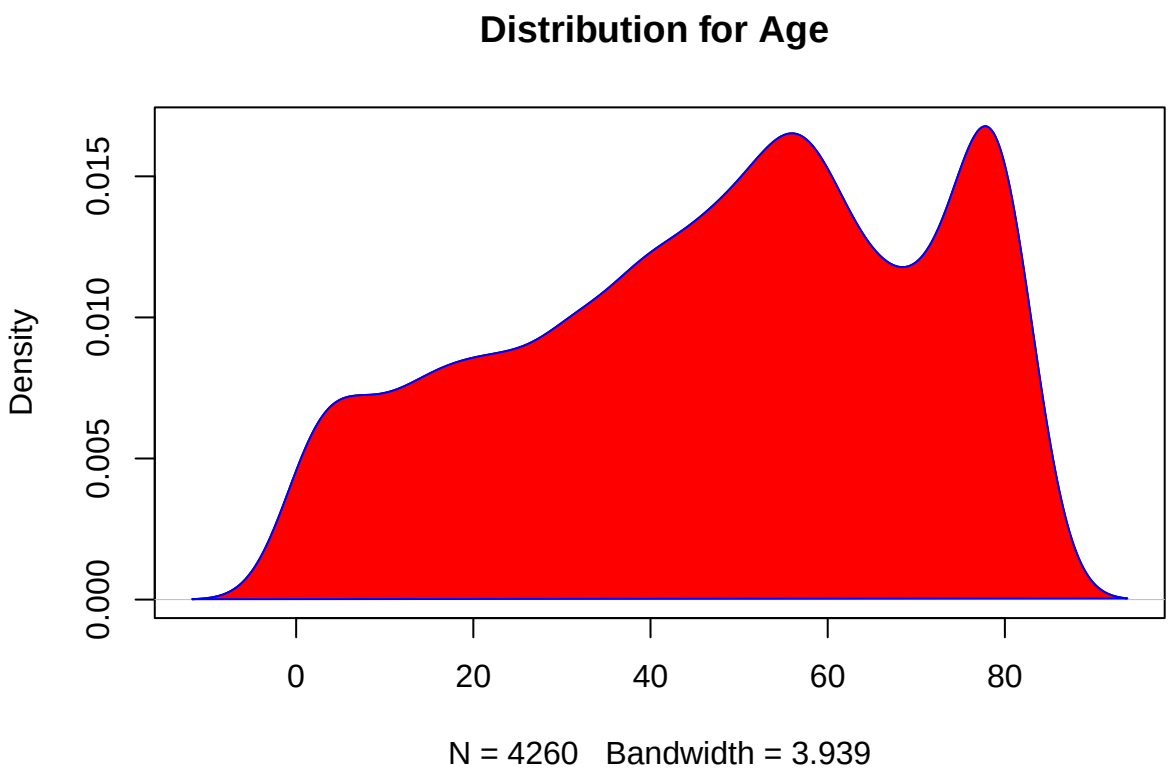
The data is highly imbalanced due to the fact the there are very few people having suffered from stroke. So to reduce the class imbalance we use oversampling techniques to replicate some data without affecting the probability distribution of predictors in the data set. To verify the probability distribution we compare the distribution plot for continuous variables before and after the balancing.

Distribution of Numerical variables in the balanced Dataset

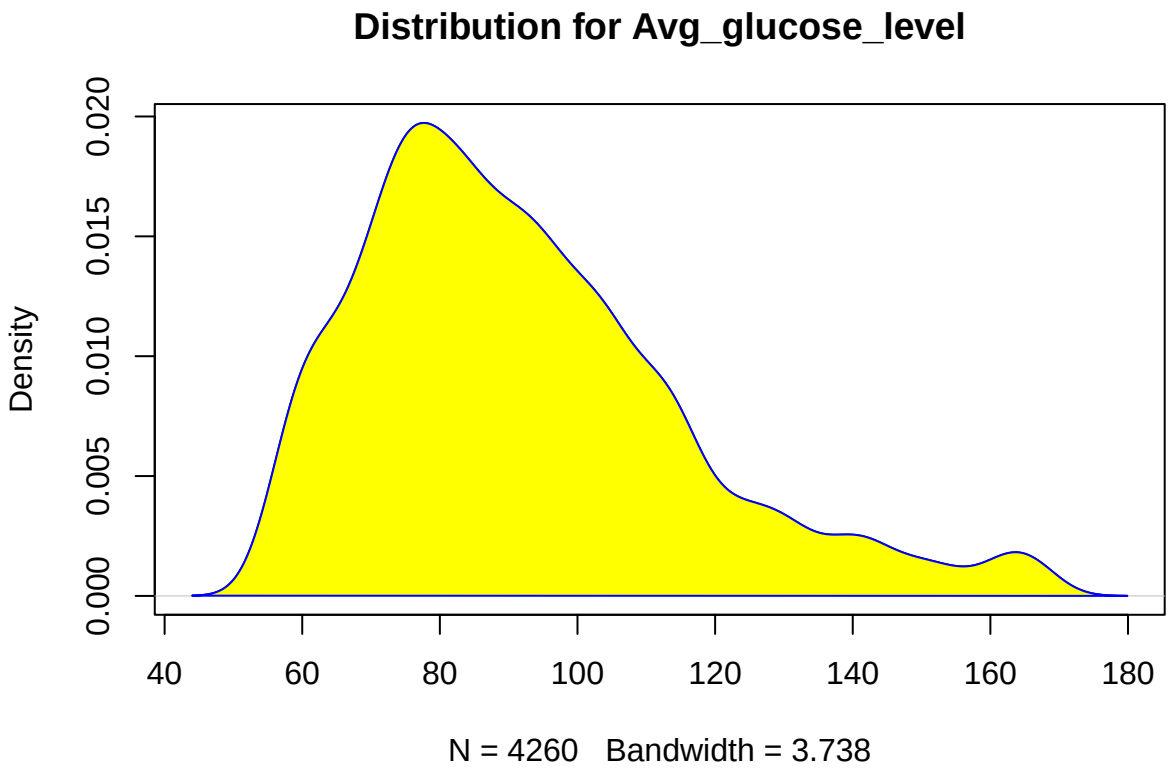
```
# distribution of balanced data

# distribution for BMI
d <- density(na.omit(data_balanced$bmi))
plot(d, main="Distribution for BMI ")
polygon(d, col="green", border="blue")
```




```
# distribution for age
d <- density(na.omit(data_balanced$age))
plot(d, main="Distribution for Age ")
polygon(d, col="red", border="blue")
```



```r
# distribution for avg_glucose_level
d <- density(na.omit(data_balanced$avg_glucose_level))
plot(d, main="Distribution for Avg_glucose_level")
polygon(d, col="yellow", border="blue")
```



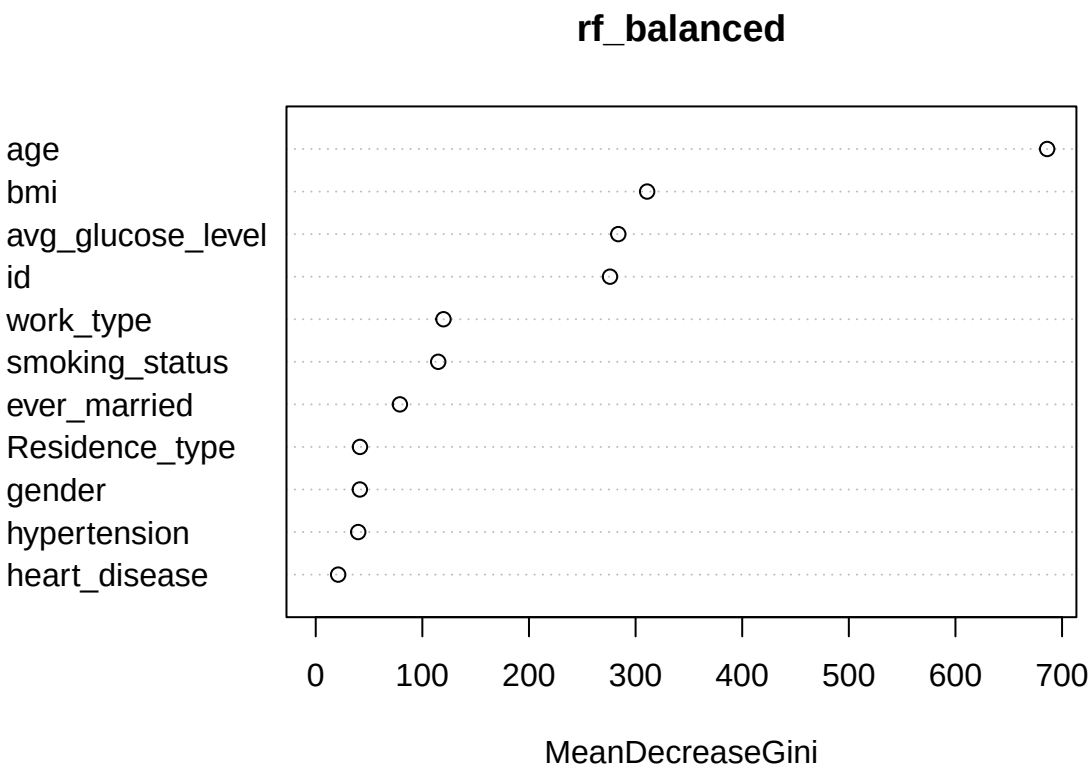
Here, we observe that the oversampling has not affected the continuous variable distribution for the model.

```r
wn = sum(data_balanced$stroke =="0")/length(data_balanced$stroke)
wy = 1

rf_balanced <- randomForest(stroke ~., data = data_balanced,
                    mtry =sqrt(ncol(data_balanced) - 1),
                    classwt = c("0"=wn, "1"=wy), ntree = 500)


# predicting the income value
yhat_rf_balanced <- predict(rf_balanced, test_data[,-14])

# accuracy
acc_rf_balanced = mean(yhat_rf_balanced == test_data$stroke)

# classification metrics
cm_rf_balanced <- confusionMatrix(yhat_rf_balanced, test_data$stroke, mode = "everything", positive="1"
cm_rf_balanced
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1263   47
##          1    5    1
```

```
##
##                Accuracy : 0.9605
##                  95% CI : (0.9485, 0.9704)
##     No Information Rate : 0.9635
##     P-Value [Acc > NIR] : 0.75
##
##                   Kappa : 0.0292
##
##  Mcnemar's Test P-Value : 1.303e-08
##
##             Sensitivity : 0.0208333
##             Specificity : 0.9960568
##          Pos Pred Value : 0.1666667
##          Neg Pred Value : 0.9641221
##               Precision : 0.1666667
##                  Recall : 0.0208333
##                      F1 : 0.0370370
##              Prevalence : 0.0364742
##          Detection Rate : 0.0007599
##    Detection Prevalence : 0.0045593
##       Balanced Accuracy : 0.5084451
##
##        'Positive' Class : 1
##
```

```
# Variable importance plot
varImpPlot(rf_balanced)
```



However, the oversampling using "ovun.sample" has not improved the model performance. The model is still predicting all observations as majority class. So, we will switch to another method for oversampling. We will make use of Synthetic Minority Oversampling Technique (SMOTE) for balancing the data.

2. Uisng SMOTE sampling technique

For Applying SMOTE on training data we need to convert all the variables to a numerical value.

```
cols<- c("gender", "hypertension", "heart_disease", "ever_married", "work_type", "Residence_type","smok
train_data[cols]<-lapply(train_data[cols], as.numeric)
test_data[cols]<-lapply(test_data[cols], as.numeric)
```

Class proportion for the balanced data

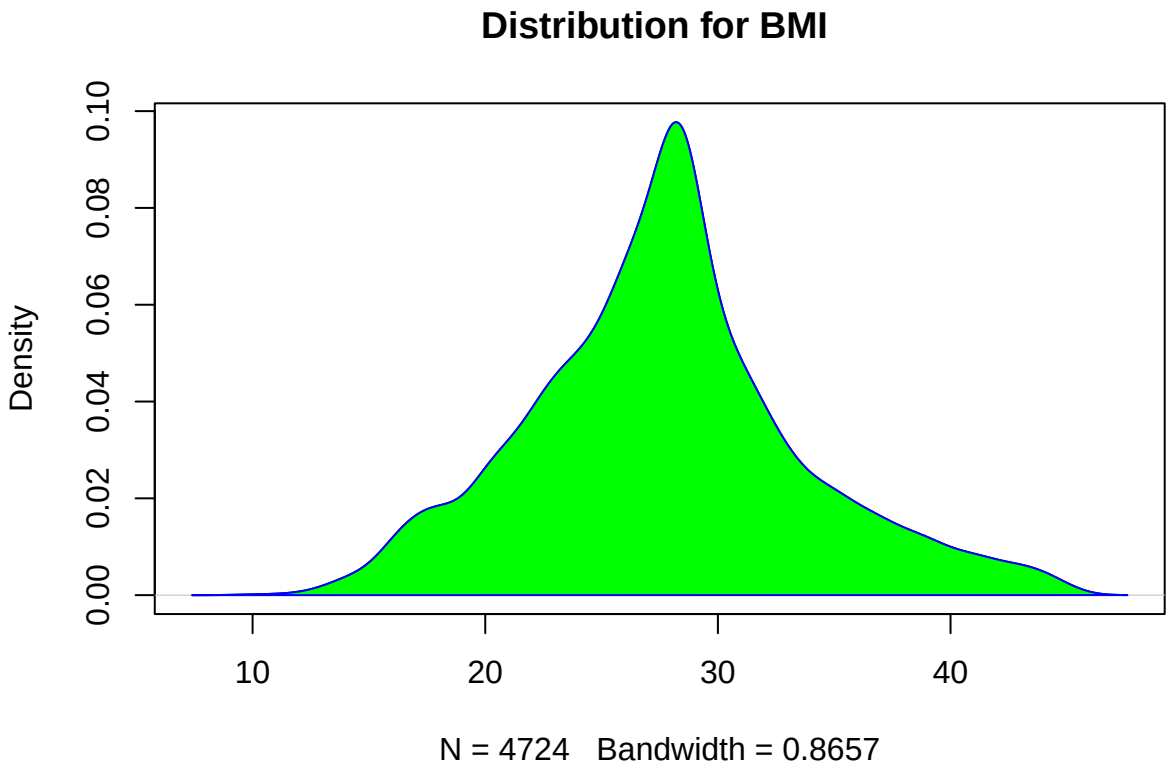
```
data_balanced_smote<-oversample(train_data, ratio = 0.6, method = "SMOTE", classAttr = "stroke")
table(data_balanced_smote$stroke)
```

```
##
##    0    1
## 2952 1772
```

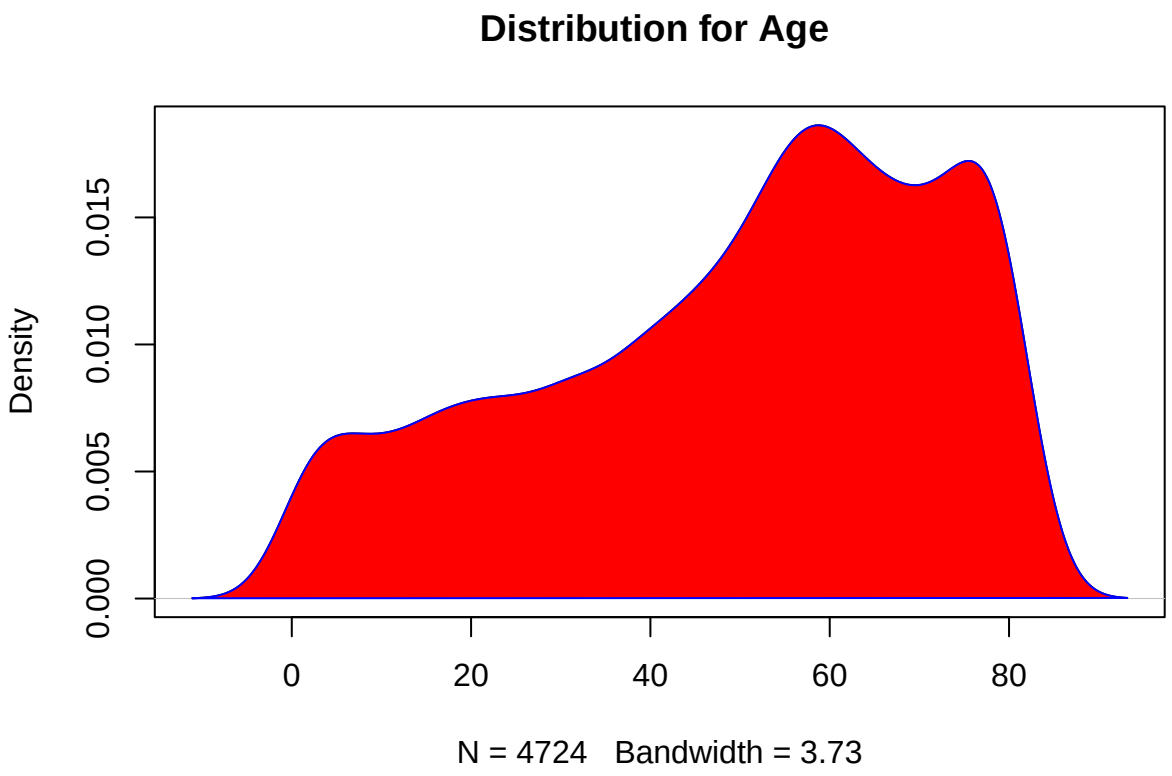
Here SMOTE adds few synthetic observations to the training data sets to increase the proportion of minority class (Class '1').

Distribution of Numerical variables after Balancing the dataset

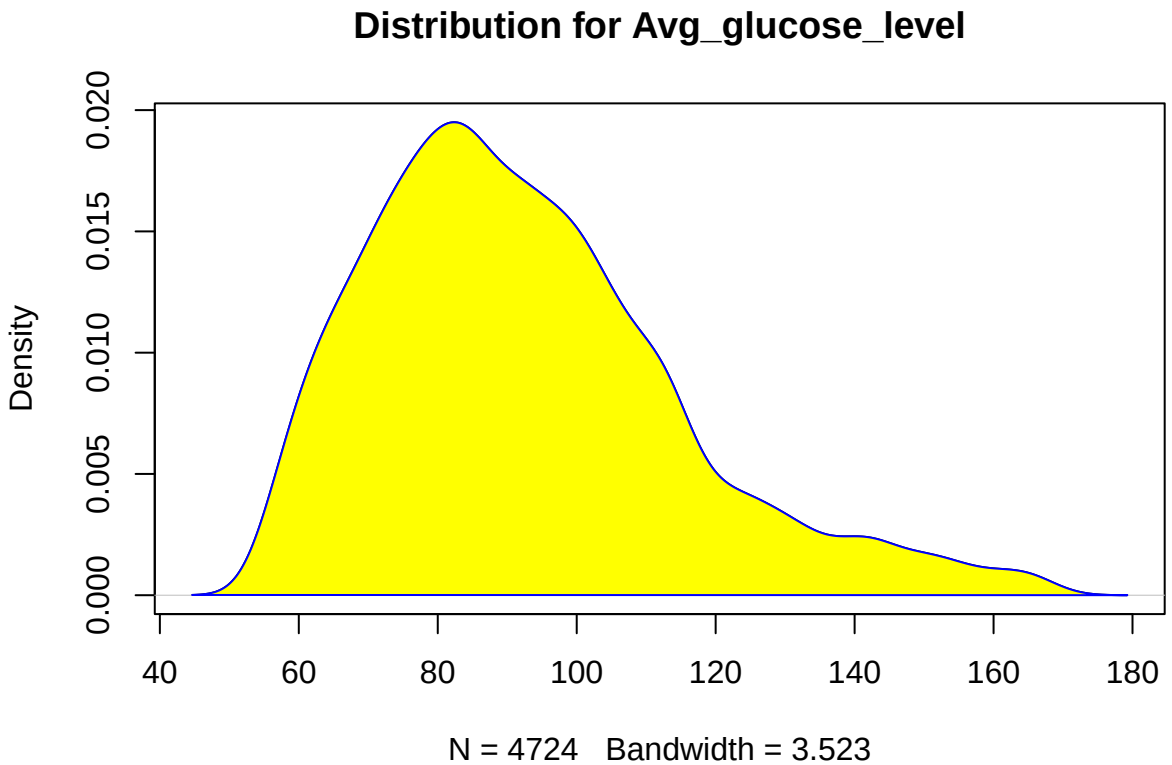
```
# distribution for BMI
d <- density(na.omit(data_balanced_smote$bmi))
plot(d, main="Distribution for BMI ")
polygon(d, col="green", border="blue")
```




```
# distribution for age
d <- density(na.omit(data_balanced_smote$age))
plot(d, main="Distribution for Age ")
polygon(d, col="red", border="blue")
```




```
# distribution for avg_glucose_level
d <- density(na.omit(data_balanced_smote$avg_glucose_level))
plot(d, main="Distribution for Avg_glucose_level")
polygon(d, col="yellow", border="blue")
```



We can observe that adding synthetic observations to balance the data does not affect the distribution for continuous

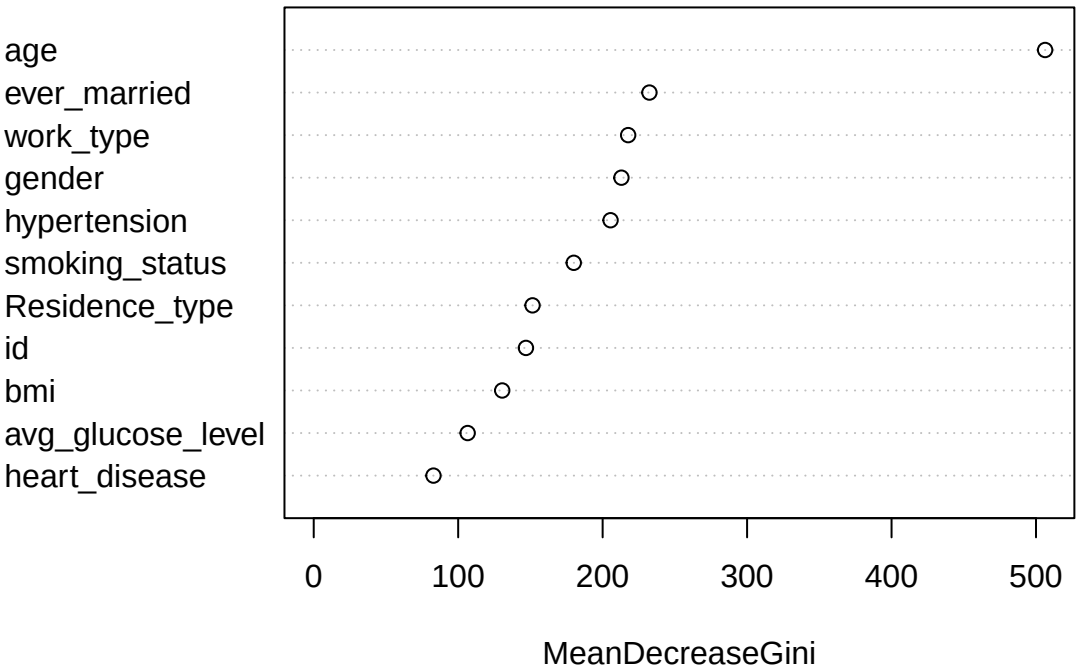
variables.

Random Forest on the Balanced Dataset

```
library(randomForest)

# modelling
data_balanced_smote$stroke<-as.factor(data_balanced_smote$stroke)
rf_smote <- randomForest(stroke ~., data = data_balanced_smote, mtry =sqrt(ncol(data_balanced_smote) - 1

# predicting the income value
yhat_rf_smote <- predict(rf_smote, test_data)

# accuracy
acc_rf_smote = mean(yhat_rf_smote == test_data$stroke)

# confusion matrix
cm_rf_smote <- confusionMatrix(yhat_rf_smote, as.factor(test_data$stroke), mode = "everything", positive
cm_rf_smote
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1268   48
##          1    0    0
##
##                Accuracy : 0.9635
##                  95% CI : (0.9519, 0.973)
##     No Information Rate : 0.9635
##     P-Value [Acc > NIR] : 0.5383
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 1.17e-11
##
##             Sensitivity : 0.00000
##             Specificity : 1.00000
##          Pos Pred Value :     NaN
##          Neg Pred Value : 0.96353
##               Precision :      NA
##                  Recall : 0.00000
##                      F1 :      NA
##              Prevalence : 0.03647
##          Detection Rate : 0.00000
##    Detection Prevalence : 0.00000
##       Balanced Accuracy : 0.50000
##
##        'Positive' Class : 1
##
```

```
# Variable importance plot
varImpPlot(rf_smote)
```



Applying other supervised machine learning models to balanced and imbalanced data.

1.

Bagging on unbalanced data

```
bagging <- randomForest(stroke ~., data = train_data, mtry =ncol(data_balanced_smote) - 1, ntree = 500)

# predicting the income value
yhat_bagging <- predict(bagging, test_data)

# accuracy
acc_bagging = mean(yhat_bagging == test_data$stroke)

# confusion matrix
cm_bagging <- confusionMatrix(yhat_bagging, as.factor(test_data$stroke), mode = "everything", positive=
cm_bagging
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1264   48
##          1    4    0
##
##                Accuracy : 0.9605
##                  95% CI : (0.9485, 0.9704)
##     No Information Rate : 0.9635
##     P-Value [Acc > NIR] : 0.75
##
##                   Kappa : -0.0056
```

```
##
##  Mcnemar's Test P-Value : 2.476e-09
##
##             Sensitivity : 0.00000
##             Specificity : 0.99685
##          Pos Pred Value : 0.00000
##          Neg Pred Value : 0.96341
##               Precision : 0.00000
##                  Recall : 0.00000
##                      F1 :     NaN
##              Prevalence : 0.03647
##          Detection Rate : 0.00000
##    Detection Prevalence : 0.00304
##       Balanced Accuracy : 0.49842
##
##        'Positive' Class : 1
##
```

Bagging on balanced data

```
data_balanced_smote$stroke<-as.factor(data_balanced_smote$stroke)
bagging_balanced <- randomForest(stroke ~., data = data_balanced_smote, mtry =ncol(data_balanced_smote)

# predicting the income value
yhat_bagging_balanced <- predict(bagging_balanced, test_data)

# accuracy
acc_bagging_balanced = mean(yhat_bagging_balanced == test_data$stroke)

# confusion matrix
cm_bagging_balanced <- confusionMatrix(yhat_bagging_balanced, as.factor(test_data$stroke), mode = "every
cm_bagging_balanced
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1263   47
##          1    5    1
##
##                Accuracy : 0.9605
##                  95% CI : (0.9485, 0.9704)
##     No Information Rate : 0.9635
##     P-Value [Acc > NIR] : 0.75
##
##                   Kappa : 0.0292
##
##  Mcnemar's Test P-Value : 1.303e-08
##
##             Sensitivity : 0.0208333
##             Specificity : 0.9960568
##          Pos Pred Value : 0.1666667
##          Neg Pred Value : 0.9641221
##               Precision : 0.1666667
```

```
##                  Recall : 0.0208333
##                      F1 : 0.0370370
##              Prevalence : 0.0364742
##          Detection Rate : 0.0007599
##    Detection Prevalence : 0.0045593
##       Balanced Accuracy : 0.5084451
##
##        'Positive' Class : 1
##
```

2.

Boosting on unbalanced data

```
train_data$stroke=as.numeric(train_data$stroke)
train_data$stroke = as.numeric(ifelse(train_data$stroke == 1, "0", "1"))

boost <- gbm(stroke~ ., data = train_data,
                        distribution = "bernoulli",
                        n.trees = 500,
                        interaction.depth = 4)

a = summary(predict(boost,test_data))
```

```
## Using 500 trees...
```

```
a
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -10.905  -6.608  -5.406  -5.157  -3.941   1.927
```

```
# prediction
yhat_boost <- ifelse(predict(boost ,newdata = test_data, n.trees =500)>a[4],1,0) #min+max/2

# accuracy
acc_boost = mean(yhat_boost == test_data$stroke)

# confusion matrix
cm_boost <- confusionMatrix(as.factor(yhat_boost), as.factor(test_data$stroke), mode = "everything", pos
cm_boost
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 714   3
##          1 554  45
##
##                Accuracy : 0.5767
##                  95% CI : (0.5495, 0.6036)
##     No Information Rate : 0.9635
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0768
##
##  Mcnemar's Test P-Value : <2e-16
```

```
##
##             Sensitivity : 0.93750
##             Specificity : 0.56309
##          Pos Pred Value : 0.07513
##          Neg Pred Value : 0.99582
##               Precision : 0.07513
##                  Recall : 0.93750
##                      F1 : 0.13910
##              Prevalence : 0.03647
##          Detection Rate : 0.03419
##    Detection Prevalence : 0.45517
##       Balanced Accuracy : 0.75030
##
##        'Positive' Class : 1
##
```

Boosting on balanced data

```
data_balanced_smote$stroke=as.numeric(data_balanced_smote$stroke)
data_balanced_smote$stroke = as.numeric(ifelse(data_balanced_smote$stroke == 1, "0", "1"))

boost_balanced <- gbm(stroke~ ., data = data_balanced_smote,
                              distribution = "bernoulli",
                              n.trees = 500,
                              interaction.depth = 4)

a = summary(predict(boost_balanced,test_data))
```

```
## Using 500 trees...
```

```
a
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -8.841  -6.023  -4.739  -4.493  -3.137   2.182
```

```
# prediction
yhat_boost_balanced <- ifelse(predict(boost_balanced ,newdata = test_data, n.trees =500)>a[4],1,0) #min

# accuracy
acc_boost_balanced = mean(yhat_boost_balanced == test_data$stroke)

# confusion matrix
cm_boost_balanced <- confusionMatrix(as.factor(yhat_boost_balanced), as.factor(test_data$stroke), mode =
cm_boost_balanced
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 707   2
##          1 561  46
##
##                Accuracy : 0.5722
##                  95% CI : (0.5449, 0.5991)
##     No Information Rate : 0.9635
##     P-Value [Acc > NIR] : 1
```

```
##
##                   Kappa : 0.0781
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.95833
##             Specificity : 0.55757
##          Pos Pred Value : 0.07578
##          Neg Pred Value : 0.99718
##               Precision : 0.07578
##                  Recall : 0.95833
##                      F1 : 0.14046
##              Prevalence : 0.03647
##          Detection Rate : 0.03495
##    Detection Prevalence : 0.46125
##       Balanced Accuracy : 0.75795
##
##        'Positive' Class : 1
##
```

3.

SVM on Unbalanced Dataset

```
library(e1071)
```

```
##
## Attaching package: 'e1071'
```

```
## The following object is masked from 'package:mltools':
##
##     skewness
```

```
train_data$stroke<- as.factor(train_data$stroke)
levels(train_data$stroke)=c(0,1)
# model fitting
svmfit <- svm (stroke ~ .,
               data = train_data,
               kernel = "linear",
               cost = 0.5,
               scale = FALSE)

# prediction
test_data$stroke<- as.factor(test_data$stroke)
levels(test_data$stroke)=c(0,1)
test.pred <- predict (svmfit , test_data)

# accuracy
acc_svm = mean(test.pred == test_data$stroke)

# confusion matrix
cm_svm <- confusionMatrix(test.pred, test_data$stroke, mode = "everything", positive="1")
cm_svm
```

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

```
##           Reference
## Prediction    0    1
##          0 1268   47
##          1    0    1
##
##                Accuracy : 0.9643
##                  95% CI : (0.9528, 0.9736)
##     No Information Rate : 0.9635
##     P-Value [Acc > NIR] : 0.4797
##
##                   Kappa : 0.0394
##
##  Mcnemar's Test P-Value : 1.949e-11
##
##             Sensitivity : 0.0208333
##             Specificity : 1.0000000
##          Pos Pred Value : 1.0000000
##          Neg Pred Value : 0.9642586
##               Precision : 1.0000000
##                  Recall : 0.0208333
##                      F1 : 0.0408163
##              Prevalence : 0.0364742
##          Detection Rate : 0.0007599
##    Detection Prevalence : 0.0007599
##       Balanced Accuracy : 0.5104167
##
##        'Positive' Class : 1
##
```

SVM on Balanced Dataset

```
library(e1071)
data_balanced_smote$stroke<- as.factor(data_balanced_smote$stroke)
levels(data_balanced_smote$stroke)=c(0,1)
set.seed (1)
# model fitting
svmfit <- svm (stroke ~ .,
               data = data_balanced_smote,
               kernel = "linear",
               cost = 0.5,
               scale = FALSE)

# prediction
test_data$stroke<- as.factor(test_data$stroke)
levels(test_data$stroke)=c(0,1)
test.pred_balanced  <- predict (svmfit , test_data)

# accuracy
acc_svm_balanced = mean(test.pred_balanced == test_data$stroke)

# confusion matrix
cm_svm_balanced <- confusionMatrix(test.pred_balanced, test_data$stroke, mode = "everything", positive=
cm_svm_balanced
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 749  30
##          1 519  18
##
##                Accuracy : 0.5828
##                  95% CI : (0.5556, 0.6096)
##     No Information Rate : 0.9635
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : -0.0058
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.37500
##             Specificity : 0.59069
##          Pos Pred Value : 0.03352
##          Neg Pred Value : 0.96149
##               Precision : 0.03352
##                  Recall : 0.37500
##                      F1 : 0.06154
##              Prevalence : 0.03647
##          Detection Rate : 0.01368
##    Detection Prevalence : 0.40805
##       Balanced Accuracy : 0.48285
##
##        'Positive' Class : 1
##
```

4.

Naive Bayes on Unbalanced Dataset by using prior = 1

```
set.seed(1)
library(e1071)

nbfit <- naiveBayes(stroke ~ ., data = train_data, prior = 1)
test_data$stroke<- as.factor(test_data$stroke)
levels(test_data$stroke)=c(0,1)

# prediction
test.pred <- predict (nbfit , test_data)

# accuracy
acc_nb = mean(test.pred == test_data$stroke)
acc_nb
```

```
## [1] 0.9004559
```

```
# confusion matrix
cm_nb <- confusionMatrix(test.pred, as.factor(test_data$stroke), mode = "everything", positive="1")
cm_nb
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1167   30
##          1  101   18
##
##                Accuracy : 0.9005
##                  95% CI : (0.883, 0.9161)
##     No Information Rate : 0.9635
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1726
##
##  Mcnemar's Test P-Value : 9.6e-10
##
##             Sensitivity : 0.37500
##             Specificity : 0.92035
##          Pos Pred Value : 0.15126
##          Neg Pred Value : 0.97494
##               Precision : 0.15126
##                  Recall : 0.37500
##                      F1 : 0.21557
##              Prevalence : 0.03647
##          Detection Rate : 0.01368
##    Detection Prevalence : 0.09043
##       Balanced Accuracy : 0.64767
##
##        'Positive' Class : 1
##
```

Naive Bayes on Balanced Dataset by using prior = 1

```
set.seed(1)
library(e1071)

nbfit_balanced <- naiveBayes ( stroke ~ ., data = data_balanced_smote, prior = 1)
test_data$stroke<- as.factor(test_data$stroke)
levels(test_data$stroke)=c(0,1)


nbfit_balanced_pred <- predict(nbfit_balanced , test_data)

# accuracy
acc_nb_balanced = mean(nbfit_balanced_pred == test_data$stroke)
acc_nb_balanced
```

```
## [1] 0.7279635
```

```
# confusion matrix
cm_nb_balanced <- confusionMatrix(test.pred, as.factor(test_data$stroke), mode = "everything", positive=
cm_nb_balanced
```

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

```
##           Reference
## Prediction    0    1
##          0 1167   30
##          1  101   18
##
##                Accuracy : 0.9005
##                  95% CI : (0.883, 0.9161)
##     No Information Rate : 0.9635
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1726
##
##  Mcnemar's Test P-Value : 9.6e-10
##
##             Sensitivity : 0.37500
##             Specificity : 0.92035
##          Pos Pred Value : 0.15126
##          Neg Pred Value : 0.97494
##               Precision : 0.15126
##                  Recall : 0.37500
##                      F1 : 0.21557
##              Prevalence : 0.03647
##          Detection Rate : 0.01368
##    Detection Prevalence : 0.09043
##       Balanced Accuracy : 0.64767
##
##        'Positive' Class : 1
##
```

5.

LDA on Unbalanced Dataset

```
lda_unbalanced <- lda(stroke ~ gender+age+hypertension+heart_disease+ever_married+work_type+Residence_ty

lda_unbalanced
```

```
## Call:
## lda(stroke ~ gender + age + hypertension + heart_disease + ever_married +
##     work_type + Residence_type + avg_glucose_level + bmi + smoking_status,
##     data = train_data)
##
## Prior probabilities of groups:
##          0          1
## 0.96187683 0.03812317
##
## Group means:
##     gender      age hypertension heart_disease ever_married work_type
## 0 1.596206 39.95672     1.070122      1.034892     1.616531  3.413957
## 1 1.581197 64.98564     1.205128      1.111111     1.854701  3.914530
##   Residence_type avg_glucose_level      bmi smoking_status
## 0       1.509824          91.42615 27.37551       2.644309
## 1       1.547009          92.90974 28.18716       2.324786
##
```

```
## Coefficients of linear discriminants:
##                              LD1
## gender             -0.121327674
## age                 0.056563473
## hypertension        0.705610351
## heart_disease       0.362433440
## ever_married       -0.700322970
## work_type          -0.124288535
## Residence_type      0.089718879
## avg_glucose_level   0.003943115
## bmi                -0.030751652
## smoking_status      0.036298572
```

```
lda_unbalanced_predict <- predict(lda_unbalanced, test_data)
lda_class <- lda_unbalanced_predict$class

table(lda_class , test_data$stroke)
```

```
##
## lda_class    0    1
##         0 1267   47
##         1    1    1
```

```
acc_lda <- mean(lda_unbalanced_predict$class == test_data$stroke)

test_data$stroke = as.factor(test_data$stroke)
cm_lda <- confusionMatrix(lda_unbalanced_predict$class, test_data$stroke, mode = "everything", positive=
cm_lda
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1267   47
##          1    1    1
##
##                Accuracy : 0.9635
##                  95% CI : (0.9519, 0.973)
##     No Information Rate : 0.9635
##     P-Value [Acc > NIR] : 0.5383
##
##                   Kappa : 0.0372
##
##  Mcnemar's Test P-Value : 8.293e-11
##
##             Sensitivity : 0.0208333
##             Specificity : 0.9992114
##          Pos Pred Value : 0.5000000
##          Neg Pred Value : 0.9642314
##               Precision : 0.5000000
##                  Recall : 0.0208333
##                      F1 : 0.0400000
##              Prevalence : 0.0364742
##          Detection Rate : 0.0007599
##    Detection Prevalence : 0.0015198
```

```
##       Balanced Accuracy : 0.5100223
##
##        'Positive' Class : 1
##
```

LDA on Balanced dataset

```
lda_balanced <- lda(stroke ~ gender+age+hypertension+heart_disease+ever_married+work_type+Residence_type

lda_balanced
```

```
## Call:
## lda(stroke ~ gender + age + hypertension + heart_disease + ever_married +
##     work_type + Residence_type + avg_glucose_level + bmi + smoking_status,
##     data = data_balanced_smote)
##
## Prior probabilities of groups:
##         0         1
## 0.6248942 0.3751058
##
## Group means:
##     gender      age hypertension heart_disease ever_married work_type
## 0 1.596206 39.95672     1.070122      1.034892     1.616531  3.413957
## 1 1.558620 65.12390     1.215960      1.117464     1.853128  3.942822
##   Residence_type avg_glucose_level      bmi smoking_status
## 0       1.509824          91.42615 27.37551       2.644309
## 1       1.544815          93.96568 28.33709       2.324067
##
## Coefficients of linear discriminants:
##                              LD1
## gender             -0.219614533
## age                 0.058954007
## hypertension        0.647375680
## heart_disease       0.154917978
## ever_married       -0.460232787
## work_type          -0.102729339
## Residence_type      0.073153127
## avg_glucose_level   0.004534197
## bmi                -0.015446074
## smoking_status      0.050630006
```

```
lda_balanced_predict <- predict(lda_balanced, test_data)
lda_class <- lda_balanced_predict$class

table(lda_class , test_data$stroke)
```

```
##
## lda_class    0    1
##         0 1017    8
##         1  251   40
```

```
acc_lda_balanced <- mean(lda_balanced_predict$class == test_data$stroke)

test_data$stroke = as.factor(test_data$stroke)
cm_lda_balanced <- confusionMatrix(lda_balanced_predict$class, test_data$stroke, mode = "everything", p
```

```
cm_lda_balanced
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1017    8
##          1  251   40
##
##                Accuracy : 0.8032
##                  95% CI : (0.7807, 0.8244)
##     No Information Rate : 0.9635
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.185
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.83333
##             Specificity : 0.80205
##          Pos Pred Value : 0.13746
##          Neg Pred Value : 0.99220
##               Precision : 0.13746
##                  Recall : 0.83333
##                      F1 : 0.23599
##              Prevalence : 0.03647
##          Detection Rate : 0.03040
##    Detection Prevalence : 0.22112
##       Balanced Accuracy : 0.81769
##
##        'Positive' Class : 1
##
```

6.

Logistic on Unbalanced Dataset

```
log_unbalanced = glm(stroke ~ gender+age+hypertension+heart_disease+ever_married+work_type+Residence_ty

summary(log_unbalanced)
```

```
##
## Call:
## glm(formula = stroke ~ gender + age + hypertension + heart_disease +
##     ever_married + work_type + Residence_type + avg_glucose_level +
##     bmi + smoking_status, family = "binomial", data = train_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.9024  -0.2914  -0.1661  -0.0977   3.6226
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -6.292320   1.113865  -5.649 1.61e-08 ***
```

```
## gender            -0.209282   0.201873  -1.037    0.300
## age                0.065888   0.006821   9.659  < 2e-16 ***
## hypertension       0.386482   0.253060   1.527    0.127
## heart_disease     -0.004285   0.330155  -0.013    0.990
## ever_married      -0.218524   0.292269  -0.748    0.455
## work_type         -0.069232   0.099803  -0.694    0.488
## Residence_type     0.076098   0.197492   0.385    0.700
## avg_glucose_level  0.004953   0.004173   1.187    0.235
## bmi               -0.016679   0.018287  -0.912    0.362
## smoking_status    -0.014211   0.093041  -0.153    0.879
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 993.94  on 3068  degrees of freedom
## Residual deviance: 828.62  on 3058  degrees of freedom
## AIC: 850.62
##
## Number of Fisher Scoring iterations: 7
```

```
log_unbalanced_pred = predict(log_unbalanced,test_data,type='response')
log_unbalanced_prob = ifelse(log_unbalanced_pred>.5,1,0)

log_unbalanced_prob <- unname(log_unbalanced_prob)
acc_lr <- mean(log_unbalanced_prob == test_data$stroke)

log_unbalanced_prob <- unname(log_unbalanced_prob)

levels(log_unbalanced_prob)=c(0,1)
# log_unbalanced_prob
# test_data$stroke

test_data$stroke<- as.factor(test_data$stroke)
levels(test_data$stroke)=c(0,1)
cm_lr <- confusionMatrix(as.factor(log_unbalanced_prob), as.factor(test_data$stroke), mode = "everythin
```

```
## Warning in confusionMatrix.default(as.factor(log_unbalanced_prob),
## as.factor(test_data$stroke), : Levels are not in the same order for reference
## and data. Refactoring data to match.
```

```
cm_lr
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1268   48
##          1    0    0
##
##                Accuracy : 0.9635
##                  95% CI : (0.9519, 0.973)
##     No Information Rate : 0.9635
##     P-Value [Acc > NIR] : 0.5383
##
```

```
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 1.17e-11
##
##             Sensitivity : 0.00000
##             Specificity : 1.00000
##          Pos Pred Value :     NaN
##          Neg Pred Value : 0.96353
##               Precision :      NA
##                  Recall : 0.00000
##                      F1 :      NA
##              Prevalence : 0.03647
##          Detection Rate : 0.00000
##    Detection Prevalence : 0.00000
##       Balanced Accuracy : 0.50000
##
##        'Positive' Class : 1
##
```

Logistic on Balanced dataset

```
log_balanced = glm(stroke ~ gender+age+hypertension+heart_disease+ever_married+work_type+Residence_type+

summary(log_balanced)
```

```
##
## Call:
## glm(formula = stroke ~ gender + age + hypertension + heart_disease +
##     ever_married + work_type + Residence_type + avg_glucose_level +
##     bmi + smoking_status, family = "binomial", data = data_balanced_smote)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2710  -0.7251  -0.2754   0.8509   3.0431
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -5.286166   0.473261 -11.170  < 2e-16 ***
## gender            -0.292429   0.082793  -3.532 0.000412 ***
## age                0.077894   0.002838  27.443  < 2e-16 ***
## hypertension       0.712943   0.117159   6.085 1.16e-09 ***
## heart_disease      0.023149   0.158732   0.146 0.884048
## ever_married      -0.264017   0.121227  -2.178 0.029415 *
## work_type         -0.034587   0.040739  -0.849 0.395884
## Residence_type     0.118441   0.080331   1.474 0.140370
## avg_glucose_level  0.006039   0.001664   3.630 0.000283 ***
## bmi               -0.001303   0.007451  -0.175 0.861183
## smoking_status     0.021380   0.038365   0.557 0.577342
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6251.0  on 4723  degrees of freedom
```

```
## Residual deviance: 4458.6  on 4713  degrees of freedom
## AIC: 4480.6
##
## Number of Fisher Scoring iterations: 5
```

```
log_balanced_pred = predict(log_balanced,test_data,type='response')
log_balanced_prob = ifelse(log_balanced_pred>.5,1,0)

log_balanced_prob <- unname(log_balanced_prob)
acc_lr_balanced <- mean(log_balanced_prob == test_data$stroke)

log_balanced_prob <- unname(log_balanced_prob)

levels(log_balanced_prob)=c(0,1)


test_data$stroke<- as.factor(test_data$stroke)
levels(test_data$stroke)=c(0,1)
cm_lr_balanced <- confusionMatrix(as.factor(log_balanced_prob), as.factor(test_data$stroke), mode = "ev
cm_lr_balanced
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1032   11
##          1  236   37
##
##                Accuracy : 0.8123
##                  95% CI : (0.7901, 0.8331)
##     No Information Rate : 0.9635
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1796
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.77083
##             Specificity : 0.81388
##          Pos Pred Value : 0.13553
##          Neg Pred Value : 0.98945
##               Precision : 0.13553
##                  Recall : 0.77083
##                      F1 : 0.23053
##              Prevalence : 0.03647
##          Detection Rate : 0.02812
##    Detection Prevalence : 0.20745
##       Balanced Accuracy : 0.79236
##
##        'Positive' Class : 1
##
```

7.

XGBoost on unbalanced data

```
x_train <- data.matrix(subset(train_data, select = -c(stroke)))
x_test <- data.matrix(subset(test_data, select = -c(stroke)))

y_train <- data.matrix(train_data$stroke)
y_test <- data.matrix(test_data$stroke)

dtrain <- xgb.DMatrix(data = x_train, label= y_train)
dtest <- xgb.DMatrix(data = x_test, label= y_test)

negative_cases <- sum(y_train == "0")
postive_cases <- sum(y_train == "1")

stroke_xgboost <- xgboost(data = dtrain,
                          nrounds =10,
                          max.depth = 5,
                          objective = "binary:logistic",
                          early_stopping_rounds = 3,
                          scale_pos_weight = negative_cases/postive_cases)
```

```
## [1]  train-logloss:0.570989
## Will train until train_logloss hasn't improved in 3 rounds.
##
## [2]  train-logloss:0.494588
## [3]  train-logloss:0.451557
## [4]  train-logloss:0.426065
## [5]  train-logloss:0.398370
## [6]  train-logloss:0.381859
## [7]  train-logloss:0.366087
## [8]  train-logloss:0.350969
## [9]  train-logloss:0.337616
## [10] train-logloss:0.330590
```

```
# prediction
pred <- predict(stroke_xgboost, dtest)
pred_class <- as.integer(pred > 0.4)

# accuracy
test <- ifelse(as.numeric(test_data$stroke) == 2 , 1, 0)
acc_xgboost = mean(pred_class == test)
acc_xgboost
```

```
## [1] 0.7393617
```

```
# confusion matrix
cm_xgboost <- confusionMatrix(as.factor(pred_class), as.factor(test), mode = "everything", positive="1")
cm_xgboost
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 940  15
##          1 328  33
##
```

```
##                Accuracy : 0.7394
##                  95% CI : (0.7147, 0.7629)
##     No Information Rate : 0.9635
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1037
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.68750
##             Specificity : 0.74132
##          Pos Pred Value : 0.09141
##          Neg Pred Value : 0.98429
##               Precision : 0.09141
##                  Recall : 0.68750
##                      F1 : 0.16137
##              Prevalence : 0.03647
##          Detection Rate : 0.02508
##    Detection Prevalence : 0.27432
##       Balanced Accuracy : 0.71441
##
##        'Positive' Class : 1
##
```

XGBoost on Balanced data

```
x_train <- data.matrix(subset(data_balanced_smote, select = -c(stroke)))
x_test <- data.matrix(subset(test_data, select = -c(stroke)))

y_train <- data.matrix(data_balanced_smote$stroke)
y_test <- data.matrix(test_data$stroke)

dtrain <- xgb.DMatrix(data = x_train, label= y_train)
dtest <- xgb.DMatrix(data = x_test, label= y_test)

negative_cases <- sum(y_train == "0")
postive_cases <- sum(y_train == "1")

stroke_xgboost <- xgboost(data = dtrain,
nrounds =10,
max.depth = 3,
objective = "binary:logistic",
early_stopping_rounds = 3,
scale_pos_weight = negative_cases/postive_cases)
```

```
## [1]  train-logloss:0.563590
## Will train until train_logloss hasn't improved in 3 rounds.
##
## [2]  train-logloss:0.486925
## [3]  train-logloss:0.429188
## [4]  train-logloss:0.387878
## [5]  train-logloss:0.355730
## [6]  train-logloss:0.328731
## [7]  train-logloss:0.299823
```

```
## [8]  train-logloss:0.283920
## [9]  train-logloss:0.264078
## [10] train-logloss:0.242324
```

```
pred <- predict(stroke_xgboost, dtest)
pred_class_balanced <- as.integer(pred > 0.4)

# accuracy
test <- ifelse(as.numeric(test_data$stroke) == 2 , 1, 0)
acc_xgboost_balanced = mean(pred_class_balanced == test)
acc_xgboost_balanced
```

```
## [1] 0.8221884
```

```
# confusion matrix
cm_xgboost_balanced <- confusionMatrix(as.factor(pred_class_balanced), as.factor(test), mode = "everythi
cm_xgboost_balanced
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1055   21
##          1  213   27
##
##                Accuracy : 0.8222
##                  95% CI : (0.8004, 0.8425)
##     No Information Rate : 0.9635
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1349
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.56250
##             Specificity : 0.83202
##          Pos Pred Value : 0.11250
##          Neg Pred Value : 0.98048
##               Precision : 0.11250
##                  Recall : 0.56250
##                      F1 : 0.18750
##              Prevalence : 0.03647
##          Detection Rate : 0.02052
##    Detection Prevalence : 0.18237
##       Balanced Accuracy : 0.69726
##
##        'Positive' Class : 1
##
```

```
cm_xgboost_balanced[["byClass"]][7]
```

```
##     F1
## 0.1875
```

5. Model Evaluation

Here we separately document the performance of selected models trained on balanced and imbalanced data.

The performance parameters taken into consideration are-

1) Accuracy
2) Balanced Accuracy
3) F1-score
4) Sensitivity

```
# accuracy
acc <- c(acc_xgboost_balanced,acc_xgboost, acc_lr_balanced, acc_lr, acc_lda_balanced,
acc_lda, acc_nb_balanced, acc_nb,
acc_svm_balanced, acc_svm, acc_bagging_balanced, acc_bagging,acc_boost_balanced, acc_boost, acc_rf_smot

# balanced accuracy
balanced_acc <- c(cm_xgboost_balanced[["byClass"]][11], cm_xgboost_balanced[["byClass"]][11],cm_lr_bala

# F1 score
f1 <- c(cm_xgboost_balanced[["byClass"]][7], cm_xgboost_balanced[["byClass"]][7],cm_lr_balanced[["byCla

# Sensitivity
sense <- c(cm_xgboost_balanced[["byClass"]][1], cm_xgboost_balanced[["byClass"]][1],cm_lr_balanced[["by

# model
model <- c("xgboost_balanced", "xgboost", "LR_balanced", "LR", "LDA_balanced", "LDA", "NB_balanced", "N

model_performance <- data.frame(model, sense, f1 ,balanced_acc, acc)
model_performance
```

```
##               model      sense         f1 balanced_acc       acc
## 1  xgboost_balanced 0.56250000 0.18750000    0.6972595 0.8221884
## 2           xgboost 0.56250000 0.18750000    0.6972595 0.7393617
## 3       LR_balanced 0.77083333 0.23052960    0.7923567 0.8123100
## 4                LR 0.00000000         NA    0.5000000 0.9635258
## 5      LDA_balanced 0.83333333 0.23598820    0.8176919 0.8031915
## 6               LDA 0.02083333 0.04000000    0.5100223 0.9635258
## 7       NB_balanced 0.37500000 0.21556886    0.6476735 0.7279635
## 8                NB 0.37500000 0.21556886    0.6476735 0.9004559
## 9      SVM_balanced 0.37500000 0.06153846    0.4828470 0.5828267
## 10              SVM 0.02083333 0.04081633    0.5104167 0.9642857
## 11 Bagging_balanced 0.02083333 0.03703704    0.5084451 0.9604863
## 12          Bagging 0.00000000        NaN    0.4984227 0.9604863
## 13   Boost_balanced 0.95833333 0.14045802    0.7579522 0.5721884
## 14            Boost 0.93750000 0.13910355    0.7502957 0.5767477
## 15         RF_smote 0.00000000         NA    0.5000000 0.9635258
## 16      RF_balanced 0.02083333 0.03703704    0.5084451 0.9604863
## 17               RF 0.00000000        NaN    0.4996057 0.9627660
```

```
model_performance <- data.frame(t(model_performance))
print(model_performance)
```

```
##                            X1         X2          X3         X4           X5
## model        xgboost_balanced    xgboost LR_balanced         LR LDA_balanced
```

```
## sense              0.56250000 0.56250000  0.77083333 0.00000000   0.83333333
## f1                 0.18750000 0.18750000  0.23052960       <NA>   0.23598820
## balanced_acc        0.6972595  0.6972595   0.7923567  0.5000000    0.8176919
## acc                 0.8221884  0.7393617   0.8123100  0.9635258    0.8031915
##                            X6         X7          X8          X9         X10
## model                     LDA NB_balanced          NB SVM_balanced         SVM
## sense              0.02083333  0.37500000 0.37500000   0.37500000 0.02083333
## f1                 0.04000000  0.21556886 0.21556886   0.06153846 0.04081633
## balanced_acc        0.5100223   0.6476735  0.6476735    0.4828470  0.5104167
## acc                 0.9635258   0.7279635  0.9004559    0.5828267  0.9642857
##                            X11        X12         X13         X14        X15
## model        Bagging_balanced    Bagging Boost_balanced      Boost   RF_smote
## sense              0.02083333 0.00000000     0.95833333 0.93750000 0.00000000
## f1                 0.03703704       <NA>     0.14045802 0.13910355       <NA>
## balanced_acc        0.5084451  0.4984227      0.7579522  0.7502957  0.5000000
## acc                 0.9604863  0.9604863      0.5721884  0.5767477  0.9635258
##                            X16        X17
## model        RF_balanced              RF
## sense         0.02083333 0.00000000
## f1            0.03703704       <NA>
## balanced_acc   0.5084451  0.4996057
## acc            0.9604863  0.9627660
```

```
colnames(model_performance) <- c('XGBoost Balanced','XGBoost","Logistic Regression Balanced","Logistic
                                 "LDA_balanced","LDA","Naive Bayes Balanced","Naive Bayes",
                                "SVM_balanced","SVM","Bagging_balanced","Bagging"
                                ,"Boosting_balanced","Boosting","Random Forest smote","Random  Forest
model_performance <- model_performance[-c(1),]

Imbalance_Models <- model_performance %>%
  dplyr::select('XGBoost','Logistic Regression','LDA","Naive Bayes","SVM","Bagging","Boosting","Random
rownames(Imbalance_Models) <- c('Sensitivity','F1-Score','Balanced Accuracy','Accuracy')

Balance_Models <- model_performance %>%
  dplyr::select('XGBoost Balanced','Logistic Regression Balanced','LDA_balanced","Naive Bayes Balanced"
colnames(Balance_Models) <- c('XGBoost','Logistic Regression','LDA","Naive Bayes","SVM","Bagging","Boos
rownames(Balance_Models) <- c('Sensitivity','F1-Score','Balanced Accuracy','Accuracy')
```

Now we have consolidated the performance metrics in tabular format

```
# performance of models on Imbalanced
print("Model performance on the imbalanced data")
```

```
## [1] "Model performance on the imbalanced data"
```

```
Imbalance_Models
```

```
##                      XGBoost Logistic Regression        LDA Naive Bayes
## Sensitivity       0.56250000          0.00000000 0.02083333  0.37500000
## F1-Score          0.18750000                <NA> 0.04000000  0.21556886
## Balanced Accuracy  0.6972595           0.5000000  0.5100223   0.6476735
## Accuracy           0.7393617           0.9635258  0.9635258   0.9004559
##                         SVM    Bagging   Boosting Random Forest
## Sensitivity       0.02083333 0.00000000 0.93750000    0.00000000
## F1-Score          0.04081633       <NA> 0.13910355          <NA>
## Balanced Accuracy  0.5104167  0.4984227  0.7502957     0.4996057
```

```
## Accuracy           0.9642857  0.9604863  0.5767477     0.9627660
```

```
# performance of models on balanced
print("Model performance on the balanced data")
```

```
## [1] "Model performance on the balanced data"
```

```
Balance_Models
```

```
##                      XGBoost Logistic Regression        LDA Naive Bayes
## Sensitivity       0.56250000          0.77083333 0.83333333  0.37500000
## F1-Score          0.18750000          0.23052960 0.23598820  0.21556886
## Balanced Accuracy  0.6972595           0.7923567  0.8176919   0.6476735
## Accuracy           0.8221884           0.8123100  0.8031915   0.7279635
##                         SVM    Bagging   Boosting Random Forest
## Sensitivity       0.37500000 0.02083333 0.95833333    0.00000000
## F1-Score          0.06153846 0.03703704 0.14045802          <NA>
## Balanced Accuracy  0.4828470  0.5084451  0.7579522     0.5000000
## Accuracy           0.5828267  0.9604863  0.5721884     0.9635258
```

After looking at the performance of all the models we inferred that sensitivity of XGBoost, Logistic Regression, SVM and LDA was good enough to be considered for stroke prediction.