National Research University Higher School of Economics

Master's Programme 'Data Analytics and Social Statistics (DASS)'

Machine Learning

Final Project: Stroke prediction

Contents

The project was prepared by ***Timofei Korovin****, DASS student*

Creation date: 10/04/2025

The last change date: 23/04/2025

1. The purpose of the project

The goal of this ML study project is to develop a model that can predict stroke using patients' medical indicators. First of all, we will identify features that are statistically correlated with stroke, then we will build several supervised models (Logistic regression, Random Forest) for the purpose of our research.

In addition, we want to understand if there is a risk group in which stroke is more common. To accomplish this task, we use unsupervised methods (k-means). At the end of the project, we will summarize how well we were able to accomplish the tasks.

2. Dataset description

According to the World Health Organization (WHO) stroke is the 2nd leading cause of death globally, responsible for approximately 11% of total deaths. This data set is used to predict whether a patient is likely to get stroke based on the input parameters like gender, age, various diseases, and smoking status. Each row in the data provides relevant information about the patient. The data set can be found by this link: https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset/data

Variables description

Below you can find the list of variables that data set contains:

1) id: unique identifier
2) gender: "Male", "Female" or "Other"
3) age: age of the patient
4) hypertension: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension
5) heart_disease: 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease
6) ever_married: "No" or "Yes"
7) work_type: "children", "Govt_jov", "Never_worked", "Private" or "Self-employed"
8) Residence_type: "Rural" or "Urban"
9) avg_glucose_level: average glucose level in blood
10) bmi: body mass index
11) smoking_status: "formerly smoked", "never smoked", "smokes" or "Unknown"*
12) stroke: 1 if the patient had a stroke or 0 if not

Overall, Data set contains 5110 observations and 12 variables (5 categorical and 7 numerical).

3. Data cleaning

Firstly, we will clean our data from some missing values and remove unnecessary columns.

```r
library(tidyverse)

stroke_data1 <- read.csv("healthcare-dataset-stroke-data.csv")

table(stroke_data1$smoking_status)
```

```
##
## formerly smoked    never smoked          smokes         Unknown
##             885            1892             789            1544
```

```r
table(stroke_data1$gender)
```

```
##
## Female   Male  Other
##   2994   2115      1
```

```r
#Variables transformation
stroke_data1$bmi <- as.numeric(stroke_data1$bmi)
stroke_data1$bmi[is.na(stroke_data1$bmi)] <- median(stroke_data1$bmi, na.rm = TRUE)
stroke_data1$smoking_status <- na_if(stroke_data1$smoking_status, "Unknown")
stroke_data1$stroke <- as.numeric(stroke_data1$stroke)


stroke_data1 <- stroke_data1 %>%
  select(-id) %>%
  filter(!is.na(smoking_status)) %>%
  filter(gender != "Other")


sum(is.na(stroke_data1$smoking_status))
```

```
## [1] 0
```

```r
sum(is.na(stroke_data1$bmi))
```

```
## [1] 0
```

Some NA values were simply deleted. This was the case with Smoking Status variable, where value “Unknown” was present. According to the author of the data set, this category simply means that this information is unavailable for the patient, which is NA in other words. Also, this category represents approximately 30% of the total data on this variable. If we introduce it as a separate category, it will create a false dependency in the model (e.g., the algorithm will start associating “Unknown” with stroke). For this reason, we decided to filter out this category.

Also, we removed the “Other” values for the variable “Gender” since the number of observations in it is only 1, which is extremely small and will not be useful for our models.

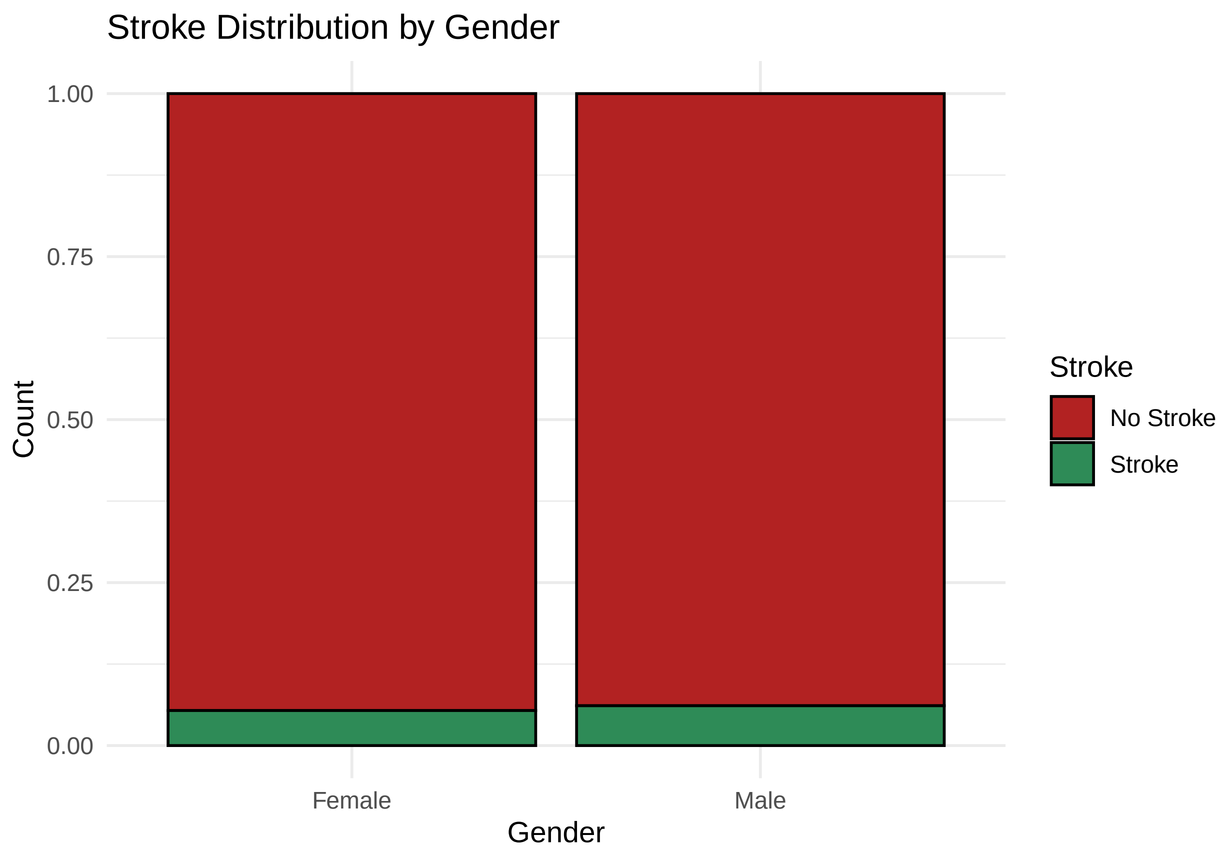
For the BMI variable, we decided to take the median for this variable to fill in the missing values. Imputing with the median is more robust than imputing with the mean, because it mitigates the effect of outliers. Thus, we will be sure that our data remains of high quality.

4. EDA

The next step of our project is exploratory data analysis. We will focus not only on the distributions of our variables, but will try to map their relationship to the target, stroke. For categorical variables (Gender, Hypertension, Heart Disease, Ever Married, Work Type, Residence Type, and Smoking status) we will build bar plots. For numeric variables (Age, BMI, and Avarage Glucose level) histograms and boxplots will be used for visual interpretation.

Stroke Distribution by Gender

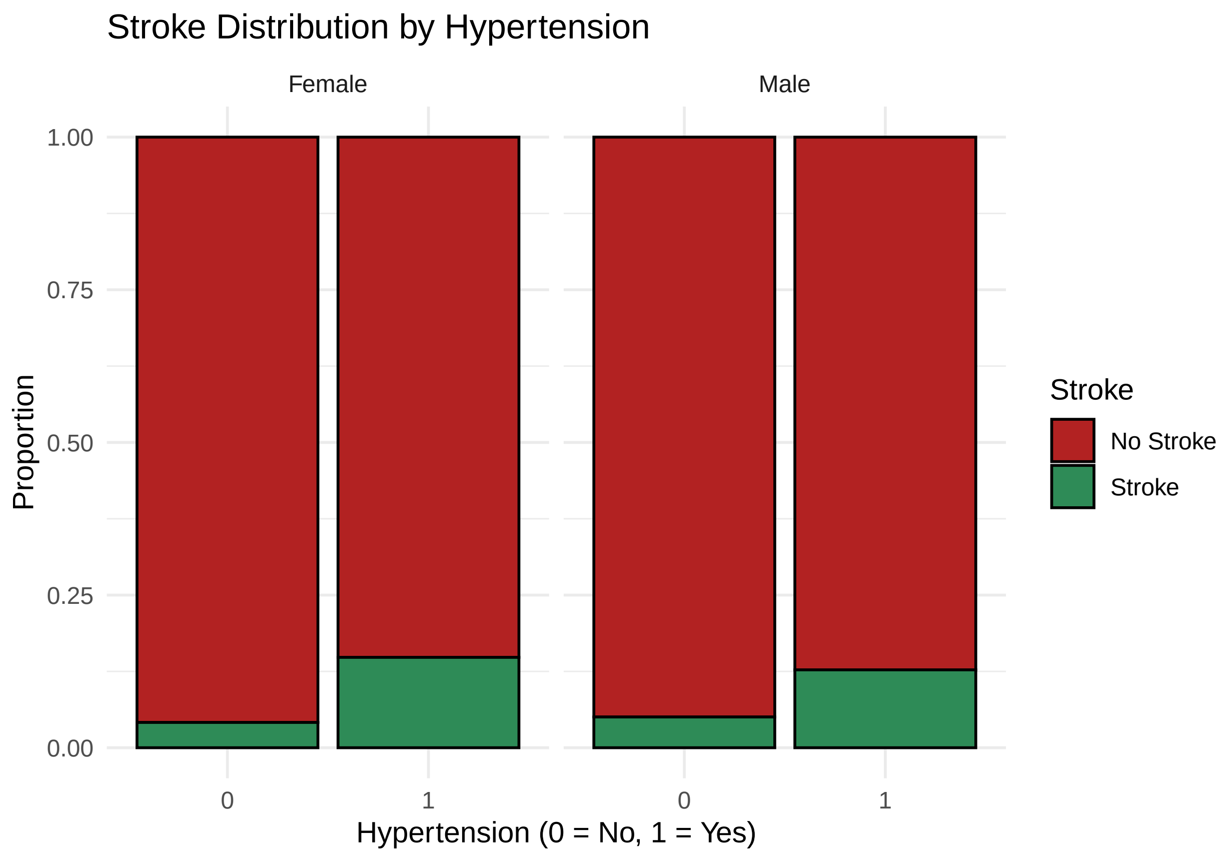
```r
ggplot(stroke_data1, aes(x = gender, fill = factor(stroke))) +
  geom_bar(position = "fill", color = "black") +
  scale_fill_manual(values = c("0" = "firebrick", "1" = "seagreen"),
                    labels = c("0" = "No Stroke", "1" = "Stroke")) +
  labs(title = "Stroke Distribution by Gender",
       x = "Gender", y = "Count", fill = "Stroke") +
  theme_minimal()
```



Based on the graph, which demonstrates the proportion of men and women with and without stroke, the distribution of strokes by Gender is relatively even, with a slight predominance of stroke cases in men. But the difference between two genders is insignificant.

Stroke Distribution by Hypertension

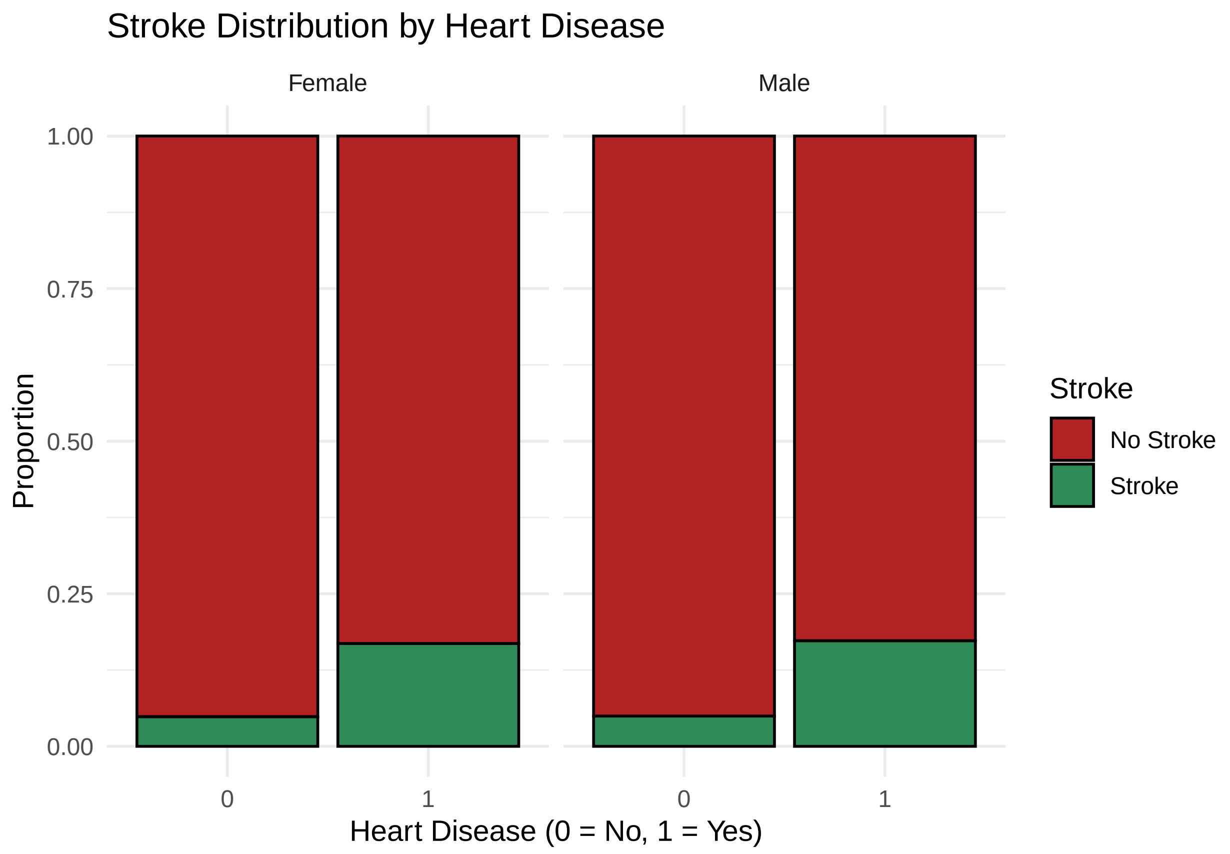
```r
ggplot(stroke_data1, aes(x = factor(hypertension), fill = factor(stroke))) +
  geom_bar(position = "fill", color = "black") +
  scale_fill_manual(values = c("0" = "firebrick", "1" = "seagreen"),
                    labels = c("0" = "No Stroke", "1" = "Stroke")) +
  labs(title = "Stroke Distribution by Hypertension",
       x = "Hypertension (0 = No, 1 = Yes)", y = "Proportion", fill = "Stroke") +
  theme_minimal() +
  facet_wrap(~gender)
```



According to the graph, in both males and females, the proportion of strokes is notably higher among those with hypertension compared to those without hypertension. This suggests a strong association between hypertension and the likelihood of stroke. Females with hypertension may have a slightly higher risk of stroke compared to males with hypertension.

Stroke Distribution by Heart Disease

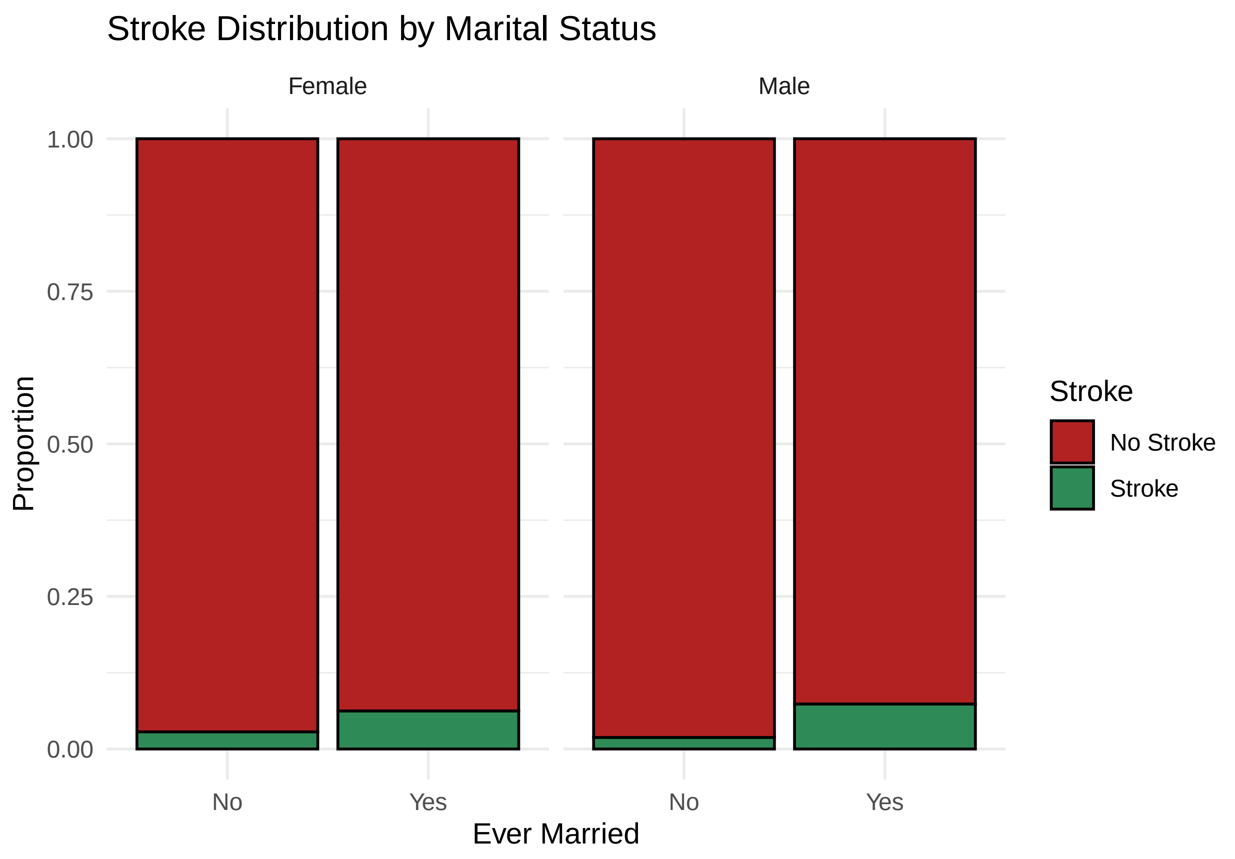
```r
ggplot(stroke_data1, aes(x = factor(heart_disease), fill = factor(stroke))) +
  geom_bar(position = "fill", color = "black") +
  scale_fill_manual(values = c("0" = "firebrick", "1" = "seagreen"),
                    labels = c("0" = "No Stroke", "1" = "Stroke")) +
  labs(title = "Stroke Distribution by Heart Disease",
       x = "Heart Disease (0 = No, 1 = Yes)", y = "Proportion", fill = "Stroke") +
  theme_minimal() +
  facet_wrap(~gender)
```



According to the graph, the proportion of strokes is higher among individuals with heart disease compared to those without heart disease, in both males and females. This indicates a positive association between heart disease and stroke incidence. The impact of heart disease on stroke risk seems relatively consistent between males and females.

Stroke Distribution by Marital Status

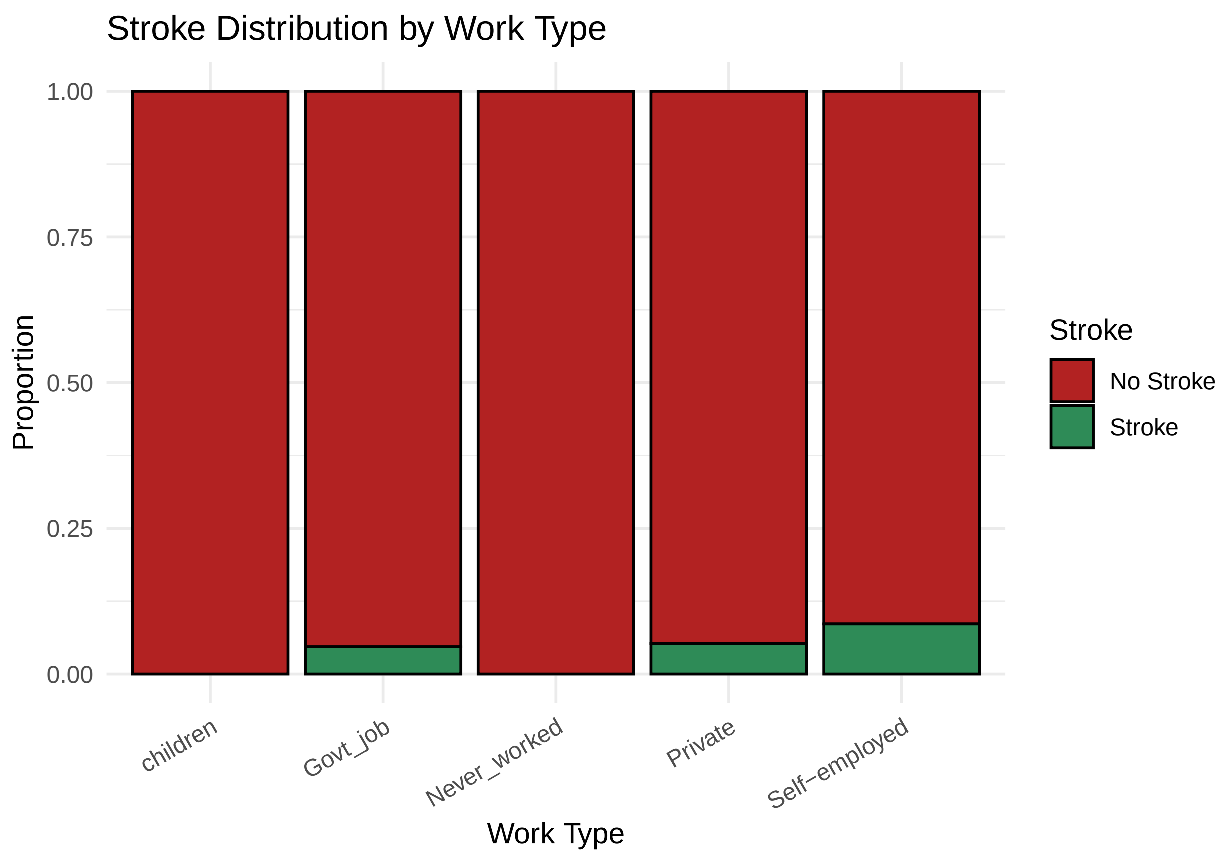
```
ggplot(stroke_data1, aes(x = ever_married, fill = factor(stroke))) +
  geom_bar(position = "fill", color = "black") +
  scale_fill_manual(values = c("0" = "firebrick", "1" = "seagreen"),
                    labels = c("0" = "No Stroke", "1" = "Stroke")) +
  labs(title = "Stroke Distribution by Marital Status",
       x = "Ever Married", y = "Proportion", fill = "Stroke") +
  theme_minimal() +
  facet_wrap(~gender)
```



The bar plot display shares of stroke and no-stroke cases by the marital status for males and females. Being married seems to have a very slight association with a higher proportion of strokes, particularly in males, but the difference is small again.

Stroke Distribution by Work Type

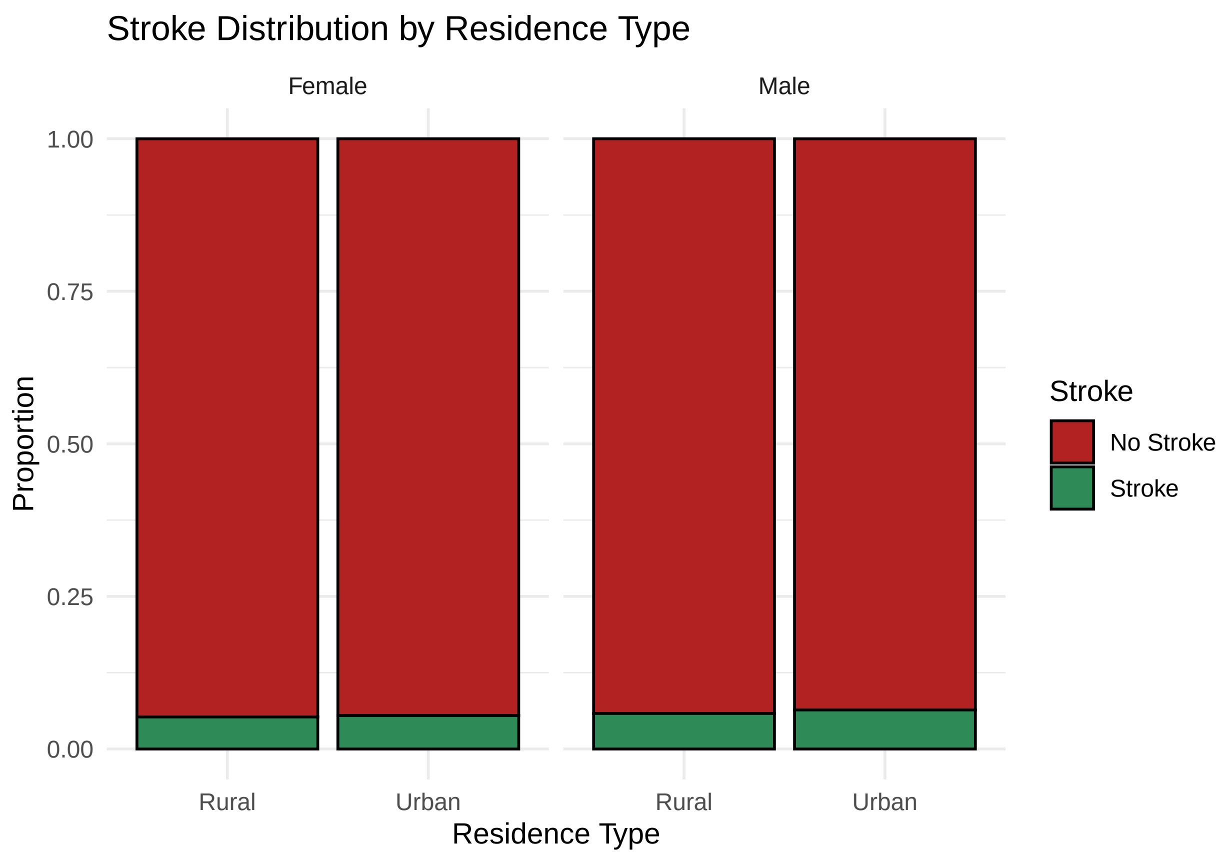
```r
ggplot(stroke_data1, aes(x = work_type, fill = factor(stroke))) +
  geom_bar(position = "fill", color = "black") +
  scale_fill_manual(values = c("0" = "firebrick", "1" = "seagreen"),
                    labels = c("0" = "No Stroke", "1" = "Stroke")) +
  labs(title = "Stroke Distribution by Work Type",
       x = "Work Type", y = "Proportion", fill = "Stroke") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 30, hjust = 1))
```



According to the graph, which demonstrate the share of stroke and no-stroke cases according to the work type of patient, self-employed patients appears to be associated with a slightly higher proportion of strokes compared to other work types (Government job, Private job). It also notable, that those who work with children or have never worked do not encounter stroke at all.

Stroke Distribution by Residence Type

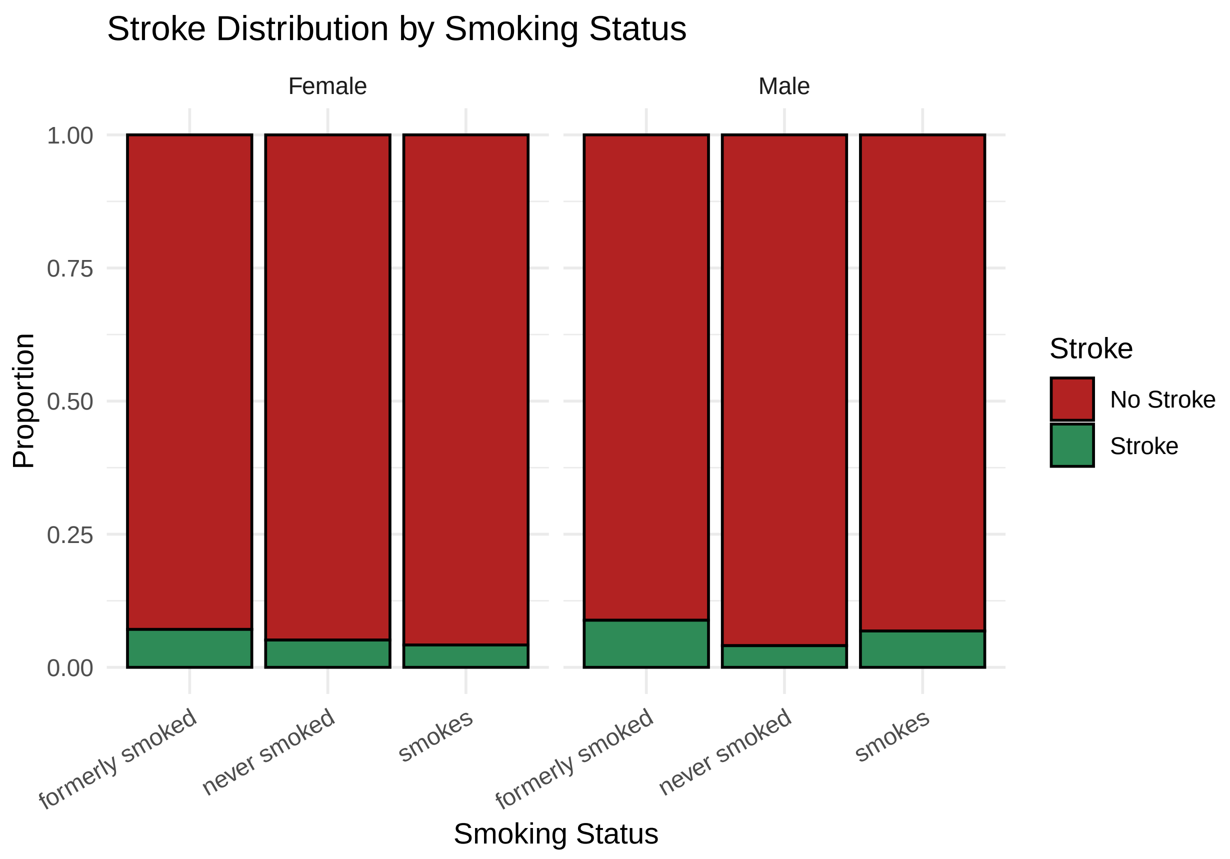
```r
ggplot(stroke_data1, aes(x = Residence_type, fill = factor(stroke))) +
  geom_bar(position = "fill", color = "black") +
  scale_fill_manual(values = c("0" = "firebrick", "1" = "seagreen"),
                    labels = c("0" = "No Stroke", "1" = "Stroke")) +
  labs(title = "Stroke Distribution by Residence Type",
       x = "Residence Type", y = "Proportion", fill = "Stroke") +
  theme_minimal() +
  facet_wrap(~gender)
```



The bar plot below represents share of stroke and no stroke cases for males and females that live in Urban and Rural areas. According to the graph, there is no difference between Residence type and stroke proportions for both genders.

Stroke Distribution by Smoking Status

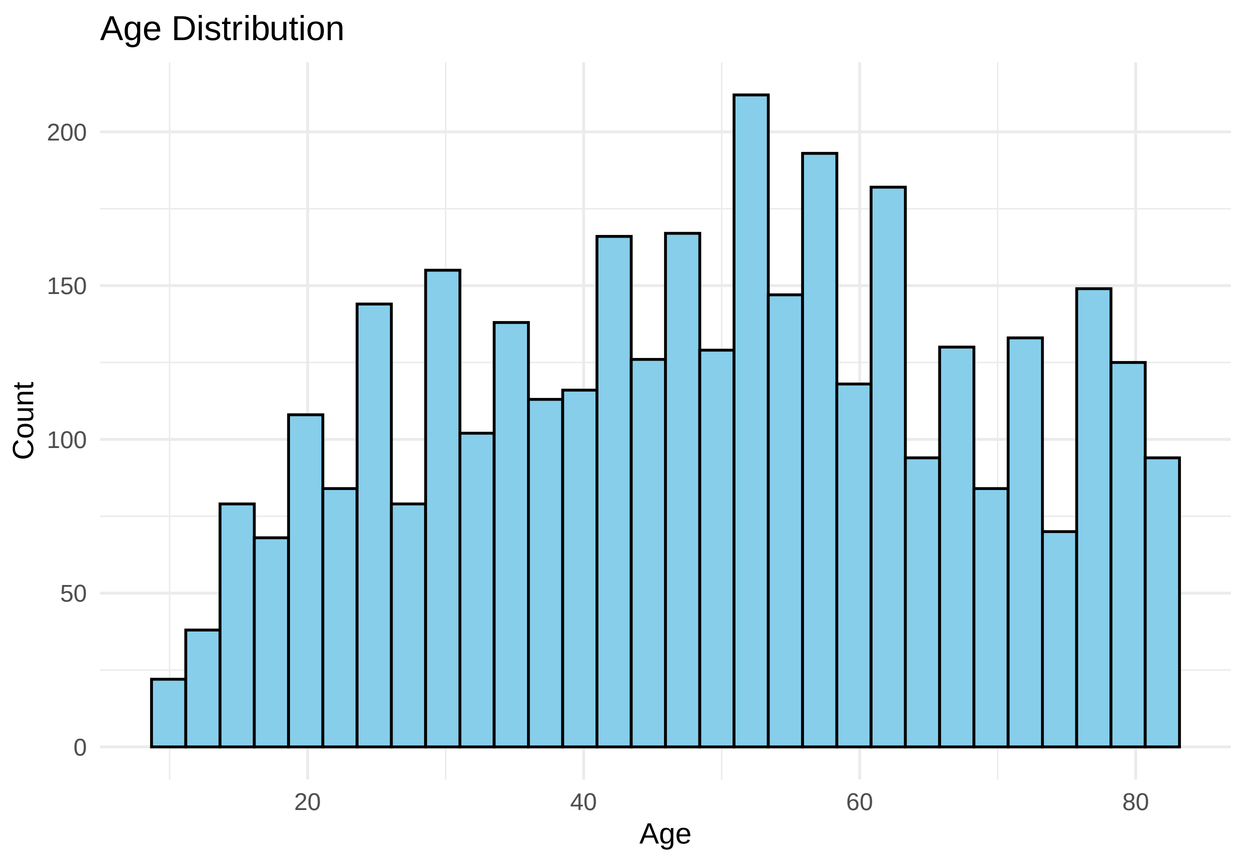
```r
ggplot(stroke_data1, aes(x = smoking_status, fill = factor(stroke))) +
  geom_bar(position = "fill", color = "black") +
  scale_fill_manual(values = c("0" = "firebrick", "1" = "seagreen"),
                    labels = c("0" = "No Stroke", "1" = "Stroke")) +
  labs(title = "Stroke Distribution by Smoking Status",
       x = "Smoking Status", y = "Proportion", fill = "Stroke") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 30, hjust = 1)) +
  facet_wrap(~gender)
```



The graph below shows the proportions of stroke and non-stroke events for male and female smokers, non-smokers, and quitters. For both females and males the formerly smoked category experience slitly proportion of strokes in comparasion to other categories. However, as it was with other factors, the difference is not that big.

Age Distribution Age

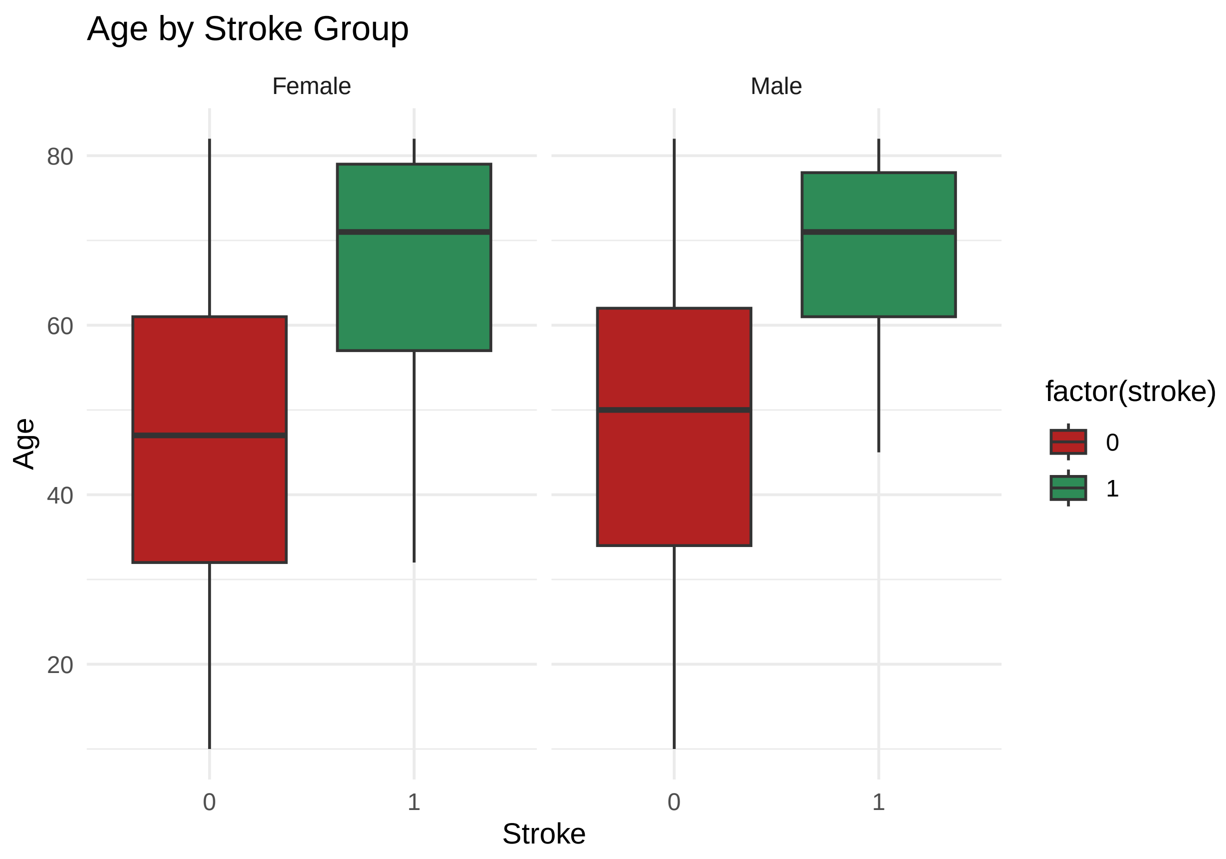
```r
ggplot(stroke_data1, aes(x = age)) +
  geom_histogram(fill = "skyblue", color = "black", bins = 30) +
  labs(title = "Age Distribution", x = "Age", y = "Count") +
  theme_minimal()
```



The distribution for Age variable looks uniform, indicating that the sample is representative. However, it seems that the 40-60 age group is more presented in the data set.

Age Distribution by Stroke Group

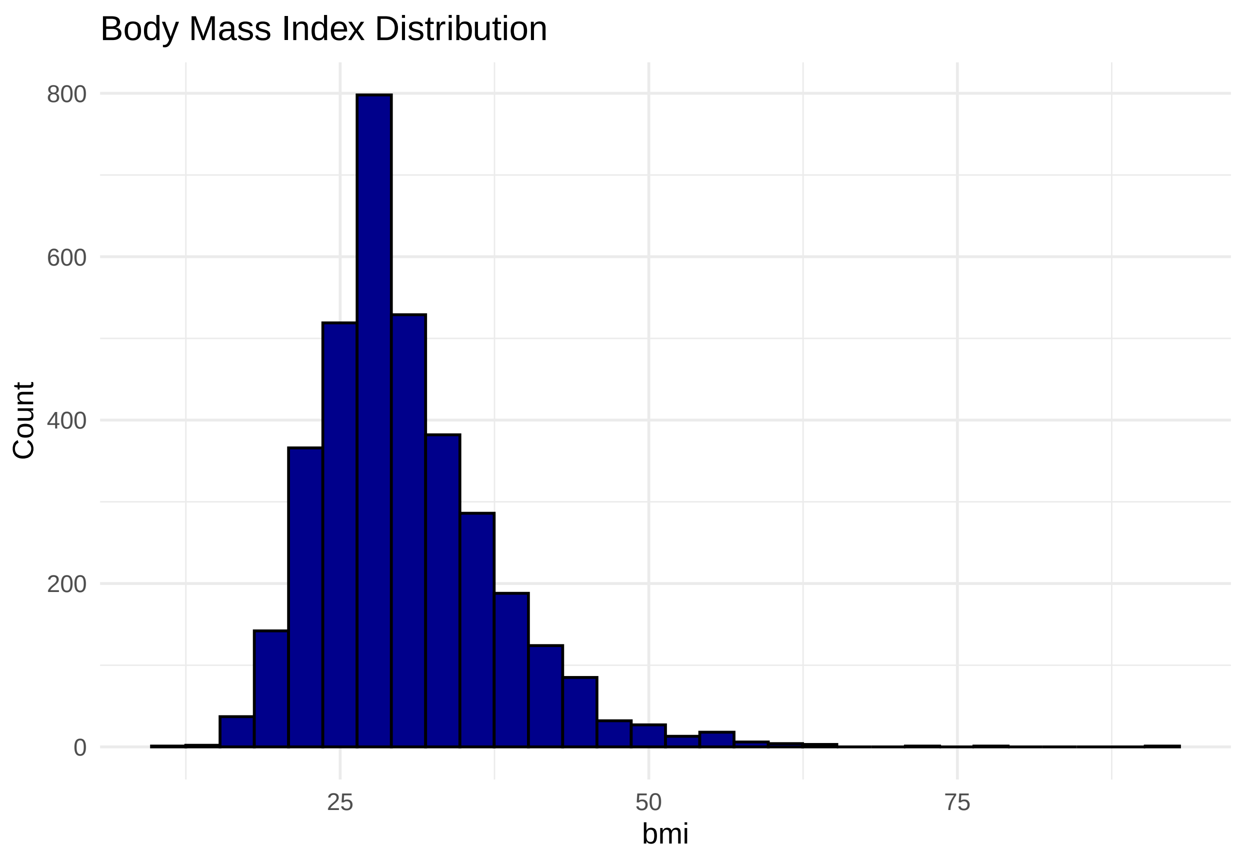
```r
ggplot(stroke_data1, aes(x = factor(stroke), y = age, fill = factor(stroke))) +
  geom_boxplot() +
  scale_fill_manual(values = c("firebrick", "seagreen")) +
  labs(title = "Age by Stroke Group", x = "Stroke", y = "Age") +
  theme_minimal() +
  facet_wrap(~gender)
```



The box plots show the distribution of age according to the presence of stroke and gender. Individuals who have had a stroke are significantly older than those who have not had a stroke, for both genders. Males and Females with stroke have a median age of about 75 years, while those who have not had a stroke are about 50 years old. Thus, there is some kind of association between age and having stroke.

BMI Distribution

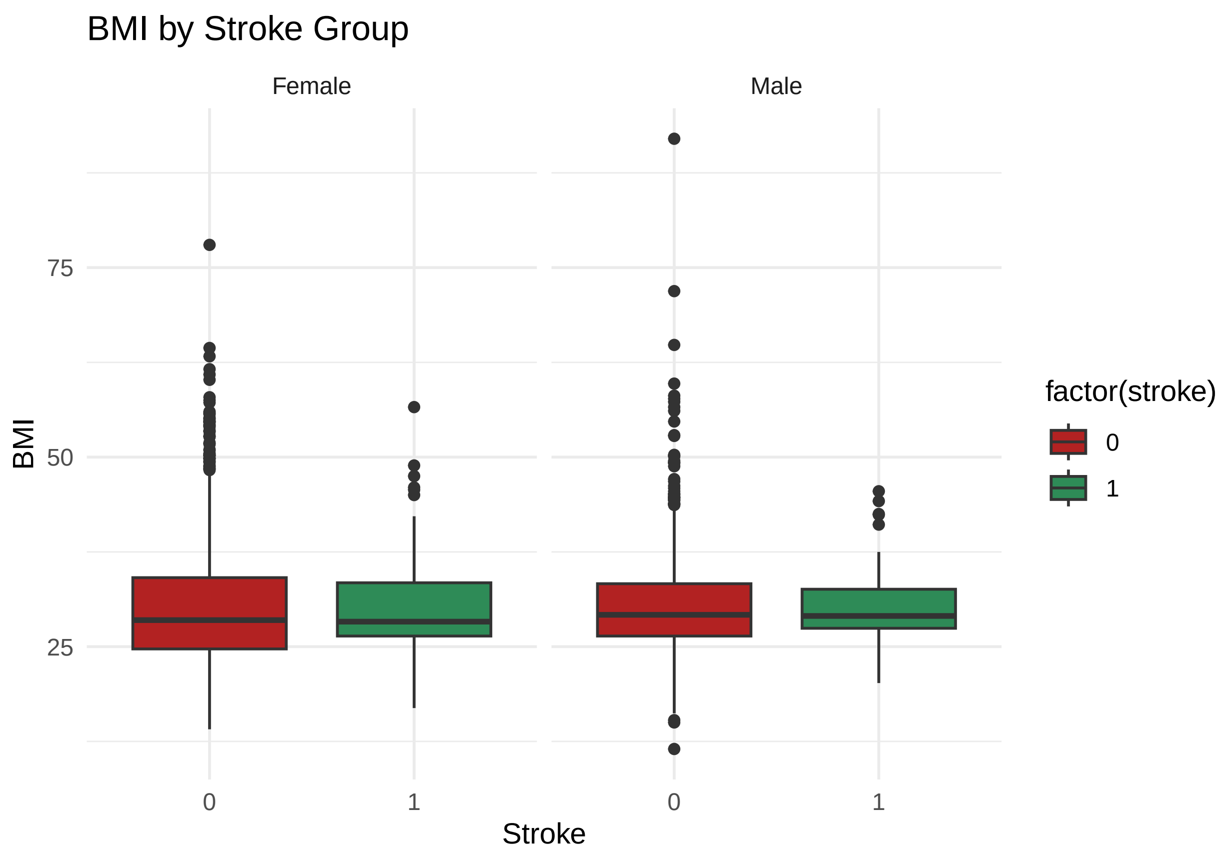
```r
ggplot(stroke_data1, aes(x = bmi)) +
  geom_histogram(fill = "darkblue", color = "black", bins = 30) +
  labs(title = "Body Mass Index Distribution", x = "bmi", y = "Count") +
  theme_minimal()
```



The distribution of BMI (Body Mass Index) is asymmetrical, with a prolonged right tail. Most values are concentrated in the range of about 20-35, with pick at around 27-30.

BMI Distribution by Stroke Group

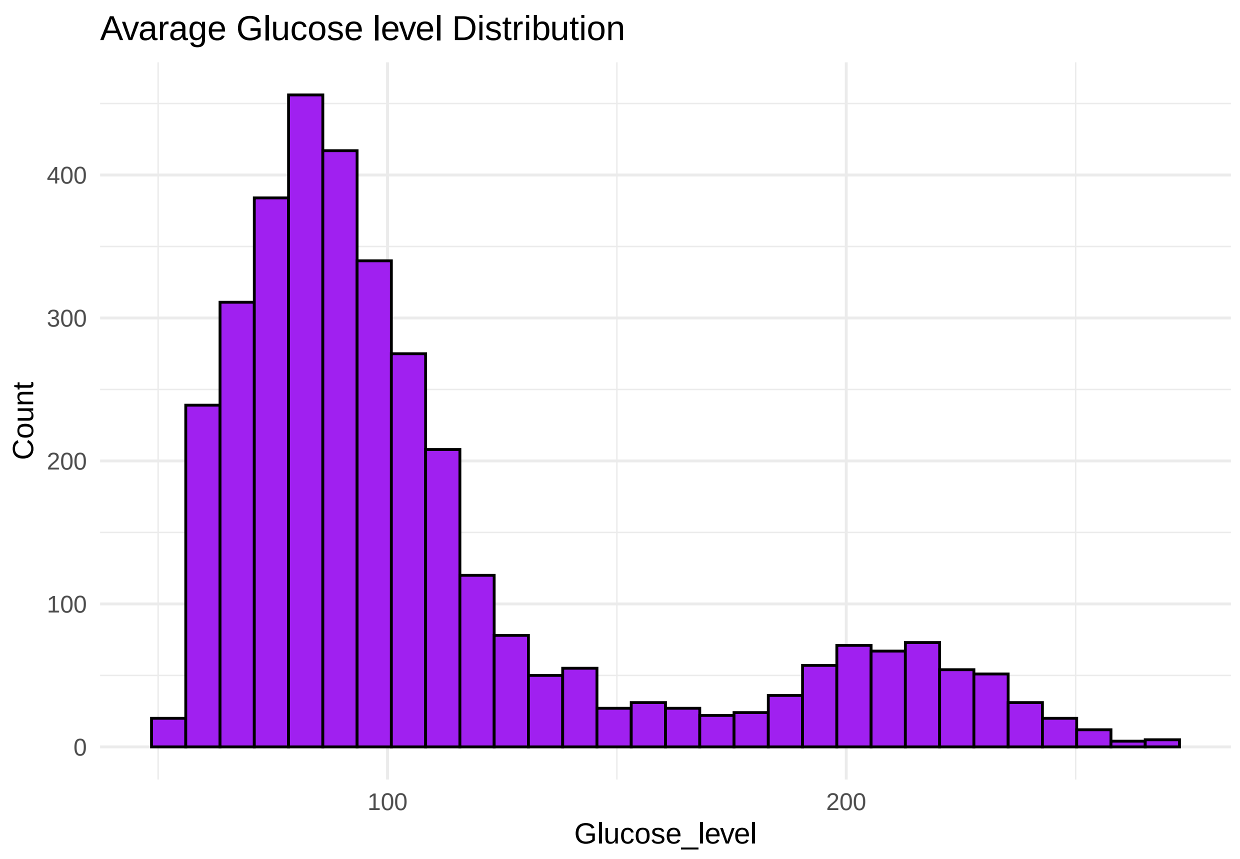
```r
ggplot(stroke_data1, aes(x = factor(stroke), y = bmi, fill = factor(stroke))) +
  geom_boxplot() +
  scale_fill_manual(values = c("firebrick", "seagreen")) +
  labs(title = "BMI by Stroke Group", x = "Stroke", y = "BMI") +
  theme_minimal() +
  facet_wrap(~gender)
```



According to the graph, that shows distribution of BMI for stroke and no-stroke groups. Overall, there is no significant difference in median BMI value for patients with stroke and without it for both genders. Also outliers are seen on the graph.

Avarage Glucose Level Distribution

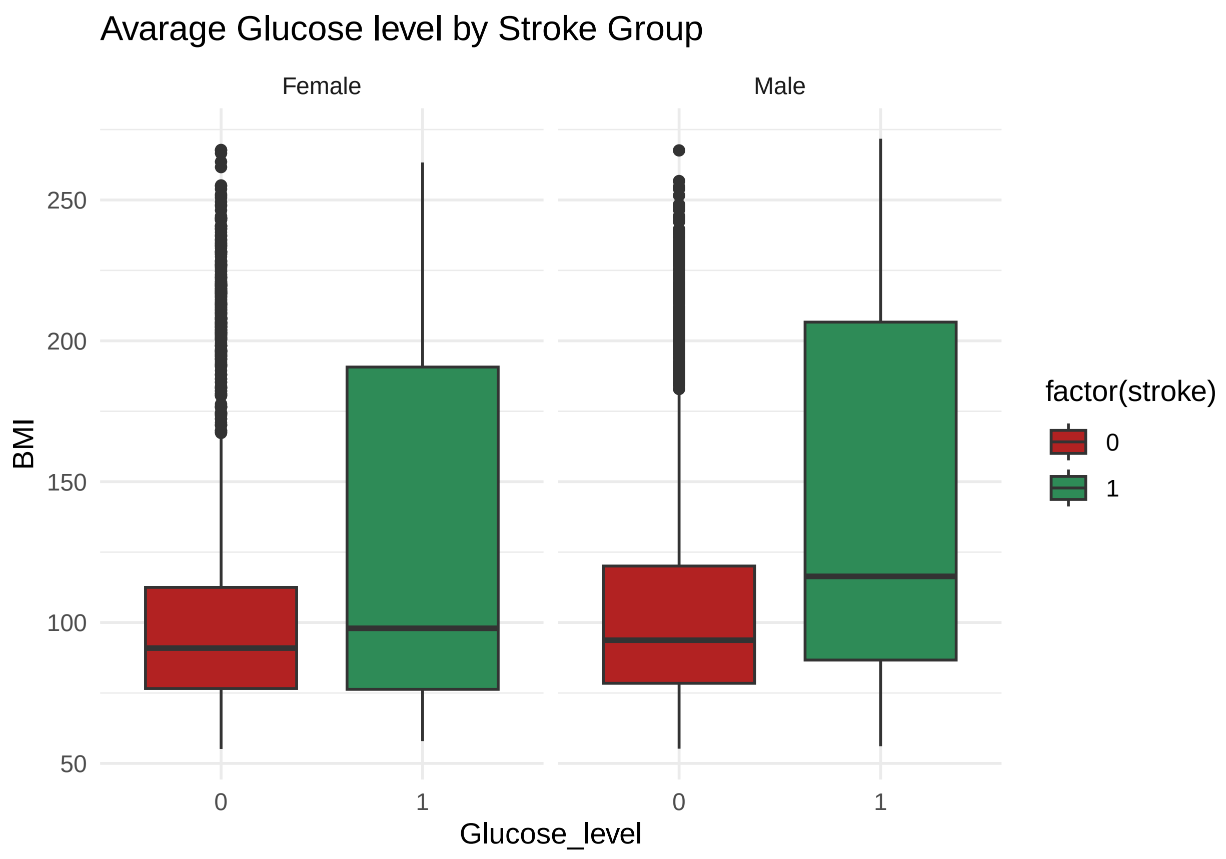
```r
ggplot(stroke_data1, aes(x = avg_glucose_level)) +
  geom_histogram(fill = "purple", color = "black", bins = 30) +
  labs(title = "Avarage Glucose level Distribution", x = "Glucose_level", y = "Count") +
  theme_minimal()
```



For the Avarage Glucose level the distribution is clearly bi-modal. The first and main pick is located between 700-100 values, and the second one between 200-220 values of glucose level.

Avarage Glucose Level Distribution by Stroke Group

```r
ggplot(stroke_data1, aes(x = factor(stroke), y = avg_glucose_level, fill = factor(stroke))) +
  geom_boxplot() +
  scale_fill_manual(values = c("firebrick", "seagreen")) +
  labs(title = "Avarage Glucose level by Stroke Group", x = "Glucose_level", y = "BMI") +
  theme_minimal() +
  facet_wrap(~gender)
```



Overall, it can be seen on the graph that for both genders the median glucose level is slightly higher for those who encountered stroke (especially for males). In addition, it is clear from the graph that our data contain some outliers.

Data Preparation for Modeling

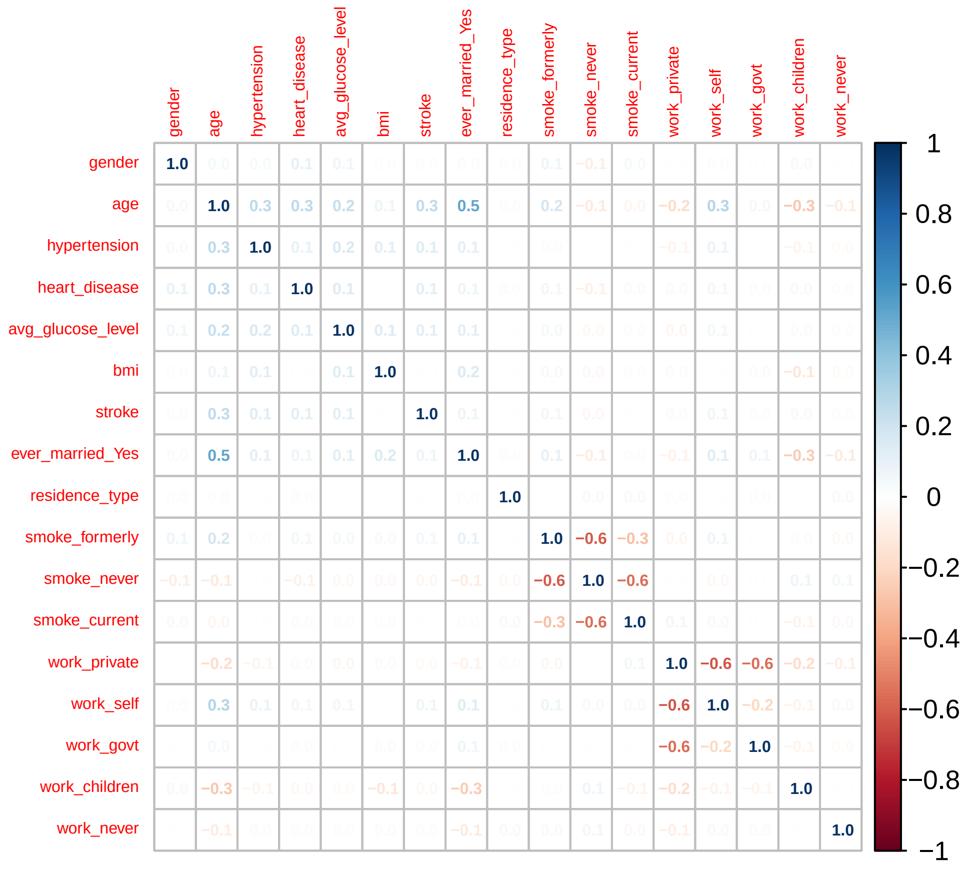
At this step of our project we transform our categorical variables using one-hot encoding. Also we apply normalization to our numerical varaibles. Finally, we plot correlation matrix to identify variables that are mostly correlated with stroke.

```
library(corrplot)
library(dplyr)
library(scales)


# 1. One-hot encoding (turning categorical variables into numeric for future
    # model building and correlation matrix)

stroke_data2 <- stroke_data1 %>%
  mutate(
    # gender: we encode it as: Female = 0, Male = 1
    gender = ifelse(gender == "Male", 1, 0),

    # ever_married: we encode it as: No = 0, Yes = 1
    ever_married_Yes = ifelse(ever_married == "Yes", 1, 0),
```

```r
    # Residence_type: we encode it as: Urban = 1, Rural = 0
    residence_type = ifelse(Residence_type == "Urban", 1, 0),

    # smoking_status: we trasnform it into three binary variables
    smoke_formerly = ifelse(smoking_status == "formerly smoked", 1, 0),
    smoke_never = ifelse(smoking_status == "never smoked", 1, 0),
    smoke_current = ifelse(smoking_status == "smokes", 1, 0),

    # work_type: we transform it into 5 binary variables
    work_private = ifelse(work_type == "Private", 1, 0),
    work_self = ifelse(work_type == "Self-employed", 1, 0),
    work_govt = ifelse(work_type == "Govt_job", 1, 0),
    work_children = ifelse(work_type == "children", 1, 0),
    work_never = ifelse(work_type == "Never_worked", 1, 0)
  ) %>% #since this columns still present in our df we should drop it
    select(-work_type, -Residence_type, -smoking_status, -ever_married)

# Correlation Matrix Building

stroke_data_cor <- stroke_data2 %>% cor(method = "pearson", use = "pairwise.complete.obs")
stroke_data_cor[, "stroke"]
```

```
##            gender               age      hypertension     heart_disease
##       0.015580112       0.250764751       0.134694998       0.129330282
## avg_glucose_level               bmi            stroke  ever_married_Yes
##       0.128773205       0.005731495       1.000000000       0.077988724
##    residence_type    smoke_formerly       smoke_never     smoke_current
##       0.007802963       0.055941560      -0.041826703      -0.007909517
##      work_private         work_self         work_govt     work_children
##      -0.023810238       0.060597823      -0.018053329      -0.034431115
##        work_never
##      -0.015388665
```

```r
corrplot(stroke_data_cor, method = "number", number.digits = 1, number.cex = 0.5, tl.cex = 0.5)
```



```r
# Data Min-Max Normalization (normalizing all the numerical variables)

stroke_data3 <- stroke_data2 %>%
  select(age, hypertension, heart_disease, avg_glucose_level, work_self, stroke) %>%
  mutate(
    age = rescale(age, to = c(0, 1)),
    hypertension = rescale(as.numeric(hypertension), to = c(0, 1)),
    avg_glucose_level = rescale(avg_glucose_level, to = c(0, 1)),
    work_self = rescale(as.numeric(work_self), to = c(0, 1)),
    stroke = as.factor(stroke)
  )
```

According to correlation matrix, 5 variables are at least weakly correlated with stroke. These are: Age, Hypertension, Heart Disease, Average glucose level, and Work Self. Thus, these variables will be used for future supervised and unsupervised modeling.

Supervised Models building

First and foremost, it is important to point out metrics we will use to evaluate our models. Since we are dealing with a highly imbalanced data (we will cover this issue below), Accuracy will not be representative in that particular case, since it's not sensitive to the minor class. For this reason, we will use Recall and Precision (to understand the trade-offs between "false negatives" and "false positives"), F1 score (for aggregated estimate on positive class). It is also important to mention that in the confusion matrices we will not see the F1-score metric, it was calculated using the following formula: F1 = 2 * (Precision * Recall) / (Precision + Recall)

Also in some cases, we will build confusion matrix for Train sample, which is not methodologically right, but it will help to diagnose overfitting.

We will try to built two type of models in our Supervised part of project: Logistic regression and Random Forest. We start with Logistic regression model.

Logistic Regression with Default parametrs

```
library(tidymodels)
library(themis)
library(glmnet)
library(yardstick)
library(ROSE)
library(pROC)
library(caret)

# Splitting our data into train and test sample

set.seed(123)

n <- nrow(stroke_data3)

train_index <- sample(1:n, size = 0.8 * n)

train_data <- stroke_data3[train_index, ]
test_data <- stroke_data3[-train_index, ]

# Using glm() function to build logistic regression
model <- glm(stroke ~ ., data = train_data, family = binomial)

# Predictions on the train sample using default threshold (0.5)
prob_predictions_train <- predict(model, newdata = train_data, type = "response")
class_predictions_train <- ifelse(prob_predictions_train > 0.5, 1, 0)

class_predictions_train <- factor(class_predictions_train, levels = c(0, 1))
train_data$stroke <- factor(train_data$stroke, levels = c(0, 1))

# Predictions on the test sample using default threshold (0.5)
prob_predictions <- predict(model, newdata = test_data, type = "response")
class_predictions <- ifelse(prob_predictions > 0.5, 1, 0)

class_predictions <- factor(class_predictions, levels = c(0, 1))
test_data$stroke <- factor(test_data$stroke, levels = c(0, 1))

# Confusion matrix for the train sample
conf_matrix_train <- confusionMatrix(class_predictions_train, train_data$stroke, positive = "1")
print(conf_matrix_train)
```

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

```
##           Reference
## Prediction    0    1
##          0 2689  163
##          1    0    0
##
##                Accuracy : 0.9428
##                  95% CI : (0.9337, 0.9511)
##     No Information Rate : 0.9428
##     P-Value [Acc > NIR] : 0.5208
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.00000
##             Specificity : 1.00000
##          Pos Pred Value :     NaN
##          Neg Pred Value : 0.94285
##              Prevalence : 0.05715
##          Detection Rate : 0.00000
##    Detection Prevalence : 0.00000
##       Balanced Accuracy : 0.50000
##
##        'Positive' Class : 1
##
```

```
# Confusion matrix for the test sample
conf_matrix_default <- confusionMatrix(as.factor(class_predictions), as.factor(test_data$stroke), posit
print(conf_matrix_default)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 674  39
##          1   0   0
##
##                Accuracy : 0.9453
##                  95% CI : (0.926, 0.9608)
##     No Information Rate : 0.9453
##     P-Value [Acc > NIR] : 0.5425
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 1.166e-09
##
##             Sensitivity : 0.0000
##             Specificity : 1.0000
##          Pos Pred Value :    NaN
##          Neg Pred Value : 0.9453
##              Prevalence : 0.0547
##          Detection Rate : 0.0000
##    Detection Prevalence : 0.0000
##       Balanced Accuracy : 0.5000
```

```
##
##        'Positive' Class : 1
##
```

Examining the results of the first model, we come to the following conclusions. First, at this stage, our model does not recognize positive class at all (presence of stroke), both in the training sample and in the test sample. However, it coups well with the negative class (absence of stroke). The reason behind it might be class imbalance, since there are too few observations for class 1 in our data set. Lets count the number of observations for positive and negative cases in stroke variable.

```r
table(stroke_data3$stroke)
```

```
##
##    0    1
## 3363  202
```

Indeed, our stroke classes are highly imbalanced. In the next iterations we will try to address this problem using different data balancing techniques (under- and over- sampling).

First, we will try oversampling, i.e. duplication of a minority class in order to improve the quality of model training.

Logistic Regression with Oversampled data

The logic behind this model is the following. Before training our model, we apply oversampling to our training sample to balance positive cases with negative. After that we will evaluate our predictions on the balanced training sample and on testing (raw) sample. Than we will calculate Recall, Precision and F1 metrics separately for train and test sample. The final step of that pipeline is to compare results of the train and test samples to evaluate how our model performance.

```r
library(tidymodels)
library(themis)
library(glmnet)
library(yardstick)
library(ROSE)
library(pROC)
library(caret)

# Splitting our data again into test and train samples

set.seed(123)
n <- nrow(stroke_data3)
train_index <- sample(1:n, size = 0.8 * n)

train_data_2 <- stroke_data3[train_index, ]
test_data_2 <- stroke_data3[-train_index, ]

# Balancing our data with oversampling method
balanced_data <- ovun.sample(stroke ~ .,
                             data = train_data_2,
                             method = "over",
                             N = 2 * table(train_data_2$stroke)[1])$data
```

```r
# Training model on the balanced data
model_balanced <- glm(stroke ~ .,
                      data = balanced_data,
                      family = binomial)

# Predictions on the Test Sample
prob_predictions_balanced <- predict(model_balanced,
                                    newdata = test_data_2,
                                    type = "response")

# Transferring probabilities into classes using default threshold (0.5)
class_predictions_balanced <- ifelse(prob_predictions_balanced > 0.5, 1, 0)


# Predictions on the Train Sample
prob_predictions_train <- predict(model_balanced,
                                  newdata = balanced_data,
                                  type = "response")

class_predictions_train <- ifelse(prob_predictions_train > 0.5, 1, 0)

# Confusion matrix for the Train Sample
conf_matrix_train <- confusionMatrix(as.factor(class_predictions_train),
                                    as.factor(balanced_data$stroke))
conf_matrix_train
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1955  515
##          1  734 2174
##
##                Accuracy : 0.7678
##                  95% CI : (0.7562, 0.779)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5355
##
##  Mcnemar's Test P-Value : 6.897e-10
##
##             Sensitivity : 0.7270
##             Specificity : 0.8085
##          Pos Pred Value : 0.7915
##          Neg Pred Value : 0.7476
##              Prevalence : 0.5000
##          Detection Rate : 0.3635
##    Detection Prevalence : 0.4593
##       Balanced Accuracy : 0.7678
##
##        'Positive' Class : 0
##
```

```r
# Confusion matrix for the Test Sample
conf_matrix_balanced <- confusionMatrix(as.factor(class_predictions_balanced),
                                        as.factor(test_data_2$stroke))
conf_matrix_balanced
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 491  10
##          1 183  29
##
##                Accuracy : 0.7293
##                  95% CI : (0.6951, 0.7616)
##     No Information Rate : 0.9453
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1528
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.7285
##             Specificity : 0.7436
##          Pos Pred Value : 0.9800
##          Neg Pred Value : 0.1368
##              Prevalence : 0.9453
##          Detection Rate : 0.6886
##    Detection Prevalence : 0.7027
##       Balanced Accuracy : 0.7360
##
##        'Positive' Class : 0
##
```

The second model, which incorporates oversampling to address class imbalance, shows a noticeable improvement in classification performance on the training set. The model demonstrates a good ability to recognize the positive class (stroke cases), achieving solid metrics: F1-score of 0.77, Recall of 0.80, and Precision of 0.74. This suggests that oversampling helped the model to better capture minority class and to reduce bias toward the majority class during training.

However, the model's generalization ability remains weak, which can be proved by a significant drop in performance on the test sample. Despite a still decent Recall of 0.74, the Precision drops to 0.14, and the F1-score drops to 0.23. This means that while the model is confident when predicting stroke cases (few false positives), it misses the majority of true stroke cases (high false negative rate).

In other words, the model tends to under-predict the minority class on the test data, leading to unacceptable outcomes in fields like healthcare, where missing positive cases can lead serious consequences.

Next, we will try to train model on undersampled data to see if there is any improvements.

Logistic regression with Undersampled data

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

```r
train_data_3 <- stroke_data3[train_index, ]
test_data_3 <- stroke_data3[-train_index, ]

balanced_data_under <- ovun.sample(stroke ~ .,
                                  data = train_data_3,
                                  method = "under",
                                  N = 2 * table(train_data_3$stroke)[2])$data

# Model
model_under <- glm(stroke ~ .,
                   data = balanced_data_under,
                   family = binomial)

# Predictions on test
prob_predictions_under <- predict(model_under,
                                 newdata = test_data_3,
                                 type = "response")

class_predictions_under <- ifelse(prob_predictions_under > 0.5, 1, 0)

# Predictions on train
prob_predictions_train_under <- predict(model_under,
                                       newdata = balanced_data_under,
                                       type = "response")

class_predictions_train_under <- ifelse(prob_predictions_train_under > 0.5, 1, 0)

# Train Conf matrix
conf_matrix_train_under <- confusionMatrix(as.factor(class_predictions_train_under),
                                          as.factor(balanced_data_under$stroke))
conf_matrix_train_under
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 114  36
##          1  49 127
##
##                Accuracy : 0.7393
##                  95% CI : (0.688, 0.7861)
##     No Information Rate : 0.5
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4785
##
##  Mcnemar's Test P-Value : 0.1931
##
##             Sensitivity : 0.6994
##             Specificity : 0.7791
##          Pos Pred Value : 0.7600
##          Neg Pred Value : 0.7216
##              Prevalence : 0.5000
##          Detection Rate : 0.3497
##    Detection Prevalence : 0.4601
##       Balanced Accuracy : 0.7393
##
##        'Positive' Class : 0
##
```

```
# Test Conf matrix
conf_matrix_under <- confusionMatrix(as.factor(class_predictions_under),
                                     as.factor(test_data_3$stroke))
conf_matrix_under
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 475  10
##          1 199  29
##
##                Accuracy : 0.7069
##                  95% CI : (0.672, 0.7401)
##     No Information Rate : 0.9453
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1366
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.7047
##             Specificity : 0.7436
##          Pos Pred Value : 0.9794
##          Neg Pred Value : 0.1272
##              Prevalence : 0.9453
##          Detection Rate : 0.6662
##    Detection Prevalence : 0.6802
##       Balanced Accuracy : 0.7242
##
##        'Positive' Class : 0
##
```

The third model, which incorporated undersampling to deal with imbalanced classes, does not show better performance. Furthermore, the results are even worse for both train and test samples than we got with oversampling. Model still preform well on the training sample (Precision 0.72, Recall 0.78, F1 0.75) but fails on the test sample (Precision 0.13, Recall 0.74, F1 0,22). The reason behind this pattern might be the data that became overfitted during the training process. However, let's try to improve our results using another model, Random Forest, which in theory should coup better with imbalanced classes.

Random Forest model

The algorithm we use to build Random Forest model is almost the same. In the first iteration we will not balance our data. Here we want to see how our model coups with the task with default parameters.

```r
library(randomForest)
library(caret)

# Splitting our data
set.seed(123)
n <- nrow(stroke_data3)
train_index <- sample(1:n, size = 0.8 * n)
train_data_4 <- stroke_data3[train_index, ]
test_data_4  <- stroke_data3[-train_index, ]

# Train model
rf_model <- randomForest(
  stroke ~ .,
  data = train_data_4,
  ntree = 1000,
  importance = TRUE
)

# Predictions on train sample
rf_prob_train <- predict(rf_model, newdata = train_data_4, type = "prob")
rf_pred_train <- ifelse(rf_prob_train[, "1"] > 0.5, "1", "0")

# Predictions on test sample
rf_prob_test <- predict(rf_model, newdata = test_data_4, type = "prob")
rf_pred_test <- ifelse(rf_prob_test[, "1"] > 0.5, "1", "0")

# Confusion matrix for train sample
conf_matrix_train <- confusionMatrix(
  factor(rf_pred_train, levels = c("0","1")),
  factor(train_data_4$stroke, levels = c("0","1")),
  positive = "1")

print(conf_matrix_train)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2689  147
##          1    0   16
##
##                Accuracy : 0.9485
##                  95% CI : (0.9397, 0.9563)
##     No Information Rate : 0.9428
##     P-Value [Acc > NIR] : 0.1043
##
##                   Kappa : 0.1703
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.09816
##             Specificity : 1.00000
##          Pos Pred Value : 1.00000
##          Neg Pred Value : 0.94817
##              Prevalence : 0.05715
##          Detection Rate : 0.00561
##    Detection Prevalence : 0.00561
##       Balanced Accuracy : 0.54908
##
##        'Positive' Class : 1
##
```

```r
# Confusion Matrix for test sample
conf_matrix_test <- confusionMatrix(
  factor(rf_pred_test, levels = c("0","1")),
  factor(test_data_4$stroke, levels = c("0","1")),
  positive = "1"
)

print(conf_matrix_test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 674  39
##          1   0   0
##
##                Accuracy : 0.9453
##                  95% CI : (0.926, 0.9608)
##     No Information Rate : 0.9453
##     P-Value [Acc > NIR] : 0.5425
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 1.166e-09
##
##             Sensitivity : 0.0000
##             Specificity : 1.0000
##          Pos Pred Value :    NaN
##          Neg Pred Value : 0.9453
##              Prevalence : 0.0547
##          Detection Rate : 0.0000
##    Detection Prevalence : 0.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 1
##
```

After first iteration building Random Forest model we got non-representative results since our model did not predict any positive cases on the test sample (as it was with first logistic model). That is why we received NaN’s and zero’s for our evaluation metrics. For this reason, we will again balance our positive cases with negative using oversampling method.

Random Forest with Oversampled Data

```r
library(randomForest)
library(caret)
library(ROSE)
library(pROC)

# Splitting our data
set.seed(123)
n <- nrow(stroke_data3)
train_index <- sample(1:n, size = 0.8 * n)

train_data_5 <- stroke_data3[train_index, ]
test_data_5 <- stroke_data3[-train_index, ]

# Oversampling
balanced_data <- ovun.sample(stroke ~ .,
                             data = train_data_5,
                             method = "over",
                             N = 2 * table(train_data_5$stroke)[1])$data

# Model training
rf_model_2 <- randomForest(stroke ~ .,
                           data = balanced_data,
                           ntree = 1000,
                           mtry = 3,
                           nodesize = 5,
                           importance = TRUE)

# Predictions on the test sample
rf_prob_test_2 <- predict(rf_model_2, newdata = test_data_5, type = "prob")
rf_pred_test_2 <- ifelse(rf_prob_test_2[, "1"] > 0.5, "1", "0")
```

```r
# Test Confusion Matrix
conf_matrix_test_2 <- confusionMatrix(
  factor(rf_pred_test_2, levels = c("0", "1")),
  factor(test_data_5$stroke, levels = c("0", "1")),
  positive = "1"
)
print(conf_matrix_test_2)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   0   1
##          0 637  35
##          1  37   4
##
##                Accuracy : 0.899
##                  95% CI : (0.8745, 0.9201)
##     No Information Rate : 0.9453
##     P-Value [Acc > NIR] : 1.0000
##
```

```
##                   Kappa : 0.0465
##
##  Mcnemar's Test P-Value : 0.9062
##
##             Sensitivity : 0.10256
##             Specificity : 0.94510
##          Pos Pred Value : 0.09756
##          Neg Pred Value : 0.94792
##              Prevalence : 0.05470
##          Detection Rate : 0.00561
##    Detection Prevalence : 0.05750
##       Balanced Accuracy : 0.52383
##
##        'Positive' Class : 1
##
```

The second random forest model started to recognize positive cases of stroke. However, the metrics for the test sample are poor again (Precision - 0.097, Recall - 0.102, F1 - 0.1), indicating that in fact our model still ignores the positive class. Most likely it happens due to the overfitting.

Random Forest with 5-Fold Cross-Validation

In this iteration we will try to use 5-Fold Cross-Validation to identify best parameters for our RF model and to deal with overfitting. In theory, Cross-validation helps prevent overfitting by training and validating the model on different subsets of the data, thus it should generalize better to unseen samples.

We will use "Caret" library for this purpose, it will automatically find the best parameters. In addition, we will still use oversampling to balance our train data set, otherwise our model will not recognize positive class again.

```r
library(randomForest)
library(caret)
library(ROSE)
library(pROC)

# Splitting our data
set.seed(123)
n <- nrow(stroke_data3)
train_index <- sample(1:n, size = 0.8 * n)

train_data_6 <- stroke_data3[train_index, ]
test_data_6 <- stroke_data3[-train_index, ]

# Oversampling
balanced_data_2 <- ovun.sample(stroke ~ .,
                             data = train_data_6,
                             method = "over",
                             N = 2 * table(train_data_6$stroke)[1])$data

# Tuning grid
tune_grid2 <- expand.grid(
  mtry = c(2, 3, 4, 5),
  splitrule = c("gini", "extratrees"),
  min.node.size = c(1, 3, 5)
)

# 5-fold cv tain control
ctrl2 <- trainControl(method = "cv",
                      number = 5,
                      classProbs = TRUE,
                      summaryFunction = twoClassSummary,
                      verboseIter = FALSE
                      )

balanced_data_2$stroke <- factor(balanced_data_2$stroke, levels = c("0", "1"))
balanced_data_2$stroke <- factor(ifelse(balanced_data_2$stroke == "1", "yes", "no"))
test_data_6$stroke <- factor(ifelse(test_data_6$stroke == "1", "yes", "no"))


# Model Training
set.seed(123)

rf_cv_model <- train(
              stroke ~ .,
              data = balanced_data_2,
              method = "ranger",
              metric = "ROC",
              trControl = ctrl2,
              tuneGrid = tune_grid2,
              num.trees = 500
              )

# Predictions on the Test sample
rf_prob_test2 <- predict(rf_cv_model, newdata = test_data_6, type = "prob")
rf_pred_test2 <- ifelse(rf_prob_test2[, "yes"] > 0.5, "yes", "no")
```

```r
# Best parametrs
print(rf_cv_model$bestTune)
```

```
##    mtry  splitrule min.node.size
## 22    5 extratrees             1
```

These are the best parameters chosen by algorithm.

```r
# Confusion Matrix
conf_matrix_test <- confusionMatrix(
  factor(rf_pred_test2, levels = c("no", "yes")),
  factor(test_data_6$stroke, levels = c("no", "yes")),
  positive = "yes"
)

conf_matrix_test
```

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

```
##           Reference
## Prediction  no yes
##        no  643  33
##        yes  31   6
##
##                Accuracy : 0.9102
##                  95% CI : (0.8868, 0.9302)
##     No Information Rate : 0.9453
##     P-Value [Acc > NIR] : 0.9999
##
##                   Kappa : 0.1105
##
##  Mcnemar's Test P-Value : 0.9005
##
##             Sensitivity : 0.153846
##             Specificity : 0.954006
##          Pos Pred Value : 0.162162
##          Neg Pred Value : 0.951183
##              Prevalence : 0.054698
##          Detection Rate : 0.008415
##    Detection Prevalence : 0.051893
##       Balanced Accuracy : 0.553926
##
##        'Positive' Class : yes
##
```

During model building process there was a problem with numerical classes (1 and 0) after applying CV to our model, it fails to work with that type of classes. For this reason we transformed them to categorical for this particular model (1 - yes, 0 - no).

Moving to the results, Cross-validation slightly improved outcomes of our model (Recall - 0.1538, Precision - 0.16, F1 - 0.15), but it still makes too many false negative and false positive predictions. The low recall indicates that most positive stroke cases remain undetected, while low precision means that many of the predicted strokes are false alarms. The F1-score, which balances both, logically remained low.

Random Forest with SMOTE and 5-fold CV

After studying several resources, we found another method, that can better balance our data. SMOTE method generate new observations for minority class instead of simply duplicating them. We still use 5-fold cross-validation to choose the best parameters for our model. Let's see if our results improved during this iteration.

```r
library(randomForest)
library(caret)
library(smotefamily)
library(pROC)
library(ranger)

# Splitting our data
set.seed(123)
n <- nrow(stroke_data3)
train_index <- sample(1:n, size = 0.8 * n)
```

```r
train_data_7 <- stroke_data3[train_index, ]
test_data_7 <- stroke_data3[-train_index, ]
train_data_7$stroke <- factor(train_data_7$stroke, levels = c("0", "1"))
train_x <- train_data_7[, setdiff(names(train_data_7), "stroke")]
train_y <- as.integer(as.character(train_data_7$stroke))

# Using smote to balance our data
smote_result <- SMOTE(X = train_x, target = train_y, K = 5, dup_size = 3)
smote_data <- smote_result$data
colnames(smote_data)[ncol(smote_data)] <- "stroke"
smote_data$stroke <- factor(ifelse(smote_data$stroke == 1, "yes", "no"))

# Tune grid
tune_grid <- expand.grid(mtry = c(2, 3, 4, 5),
                         splitrule = c("gini", "extratrees"),
                         min.node.size = c(1, 3, 5)
                         )

# 5-fold cv
ctrl <- trainControl(method = "cv",
                     number = 5,
                     classProbs = TRUE,
                     summaryFunction = twoClassSummary,
                     verboseIter = FALSE)


test_data_7$stroke <- factor(ifelse(test_data_7$stroke == "1", "yes", "no"))

# Model training
set.seed(123)
rf_cv_model <- train(stroke ~ .,
                     data = smote_data,
                     method = "ranger",
                     metric = "ROC",
                     trControl = ctrl,
                     tuneGrid = tune_grid,
                     num.trees = 500)

# Predictions on the test sample
rf_prob_test <- predict(rf_cv_model, newdata = test_data_7, type = "prob")
rf_pred_test <- ifelse(rf_prob_test[, "yes"] > 0.5, "yes", "no")
```

```r
# Best parameters found using CV
print(rf_cv_model$bestTune)
```

```
##    mtry  splitrule min.node.size
## 22    5 extratrees             1
```

```r
# Confusion Matrix
conf_matrix_test <- confusionMatrix(
  factor(rf_pred_test, levels = c("no", "yes")),
  factor(test_data_7$stroke, levels = c("no", "yes")),
  positive = "yes"
)

conf_matrix_test
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  no yes
##        no  630  29
##        yes  44  10
##
##                Accuracy : 0.8976
##                  95% CI : (0.873, 0.9189)
##     No Information Rate : 0.9453
##     P-Value [Acc > NIR] : 1.0000
##
##                   Kappa : 0.1618
##
##  Mcnemar's Test P-Value : 0.1013
##
##             Sensitivity : 0.25641
##             Specificity : 0.93472
##          Pos Pred Value : 0.18519
##          Neg Pred Value : 0.95599
##              Prevalence : 0.05470
##          Detection Rate : 0.01403
##    Detection Prevalence : 0.07574
##       Balanced Accuracy : 0.59556
##
##        'Positive' Class : yes
##
```

SMOTE technique improved the results of our model, however, it still struggles with detecting positive cases. Overall, we got a Recall of 0.256, Precision of 0.185, and an F1-score of 0.215, which are moderate improvements compared to the previous model. This indicates that the model is now identifying more true stroke cases (higher recall), but still produces a significant number of false positives (low precision).

Results of the Supervised Modeling section

The key takeaways we can make from this analysis is that all of the chosen supervised models poorly predict target variable on the test sample (when we are not balancing our positive cases with negative). For now, since we have very little experience in ML we tried to balance our data with over- and under- sampling, as well as SMOTE method to change the efficiency of our models. However, it did not go as planned. After several iterations on balancing data we still encountered poor scores. The dilemma is that without data balancing our models do not recognize positive stroke cases at all, and with it they make too many errors to call such models at least acceptable. At this stage, we do not know how else we can improve our models.

Another problem may be the independent variables themselves, that potentially weakly explain the target variable, which results in a low rates of prediction. Probably lifestyle-type of data is not the best choice of predicting a stroke.

To summarize, there are two possible problems with our models: either overfitting or inappropriate independent variables.

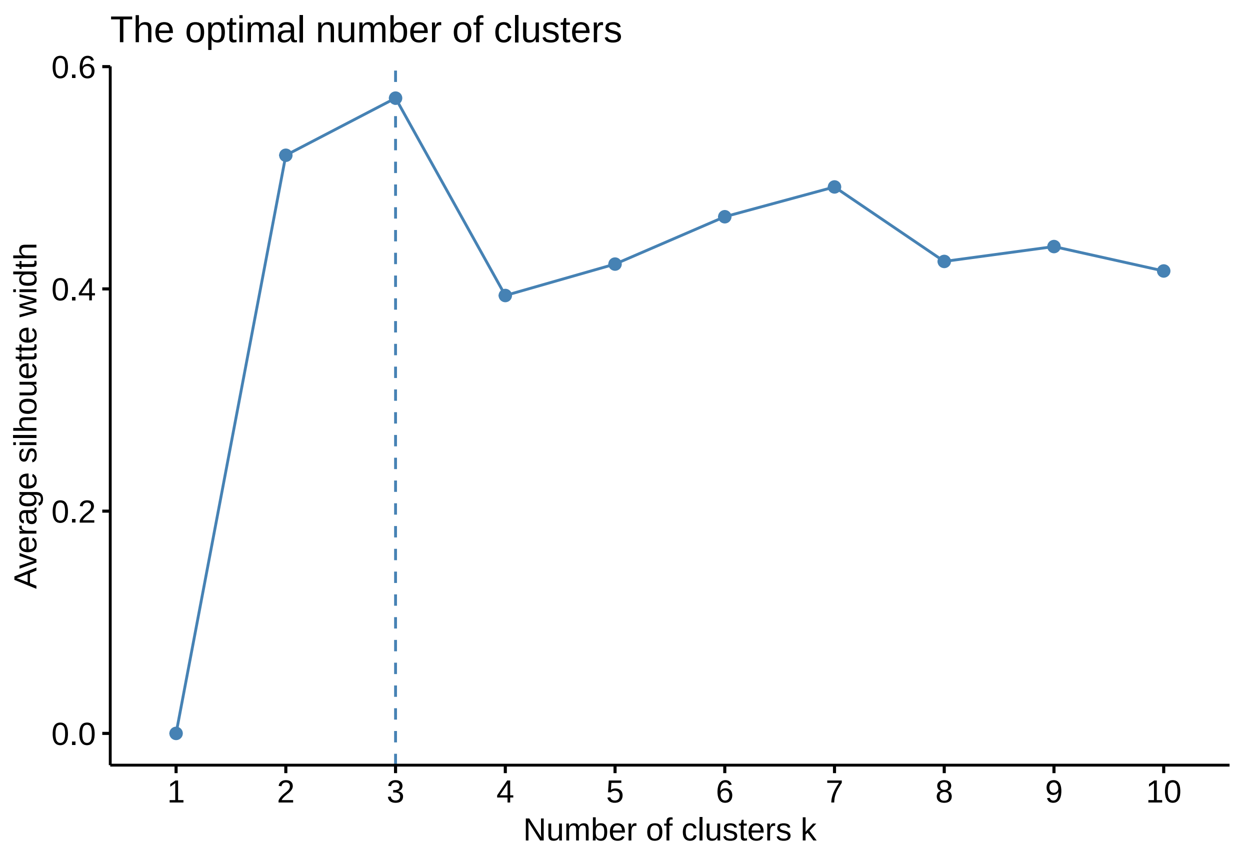
Unsupervised Modeling

To address our second question (we want to identify risk groups for stroke), we will use k-means method. Firstly, we will identify clusters using the same variables that was used in the previous part. After that we will calculate the share of patients with stoke for each cluster. One we find the cluster with the biggest share of stroke we will calculate averages for each attribute to understand which factors can potentially lead to the stroke.

K-means clustering model

Firstly, we will exclude our target variable and use Silhouette method to indetify optimal number of clusters we should use given our data.

```r
library(factoextra)
library(cluster)
library(dplyr)

data_kmeans <- stroke_data3 %>% select(-stroke)

library(factoextra)

fviz_nbclust(data_kmeans, kmeans, method = "silhouette") +
  labs(title = "The optimal number of clusters")
```



According to the graph the optimal number of clusters is 3.Moving forward, lets build k-means model and interpret the results of the clustering.

```r
set.seed(123)

kmeans_result <- kmeans(data_kmeans, centers = 3, nstart = 1)

clustered_data <- stroke_data3 %>%
  mutate(cluster = as.factor(kmeans_result$cluster))

# stroke distribution by clusters
print("Stroke distribution by clusters")
```

```
## [1] "Stroke distribution by clusters"
```

```r
table(clustered_data$cluster, clustered_data$stroke)
```

```
##
##        0    1
##   1  189   39
##   2 2622  113
##   3  552   50
```

```r
print("Stroke shares by clusters")
```

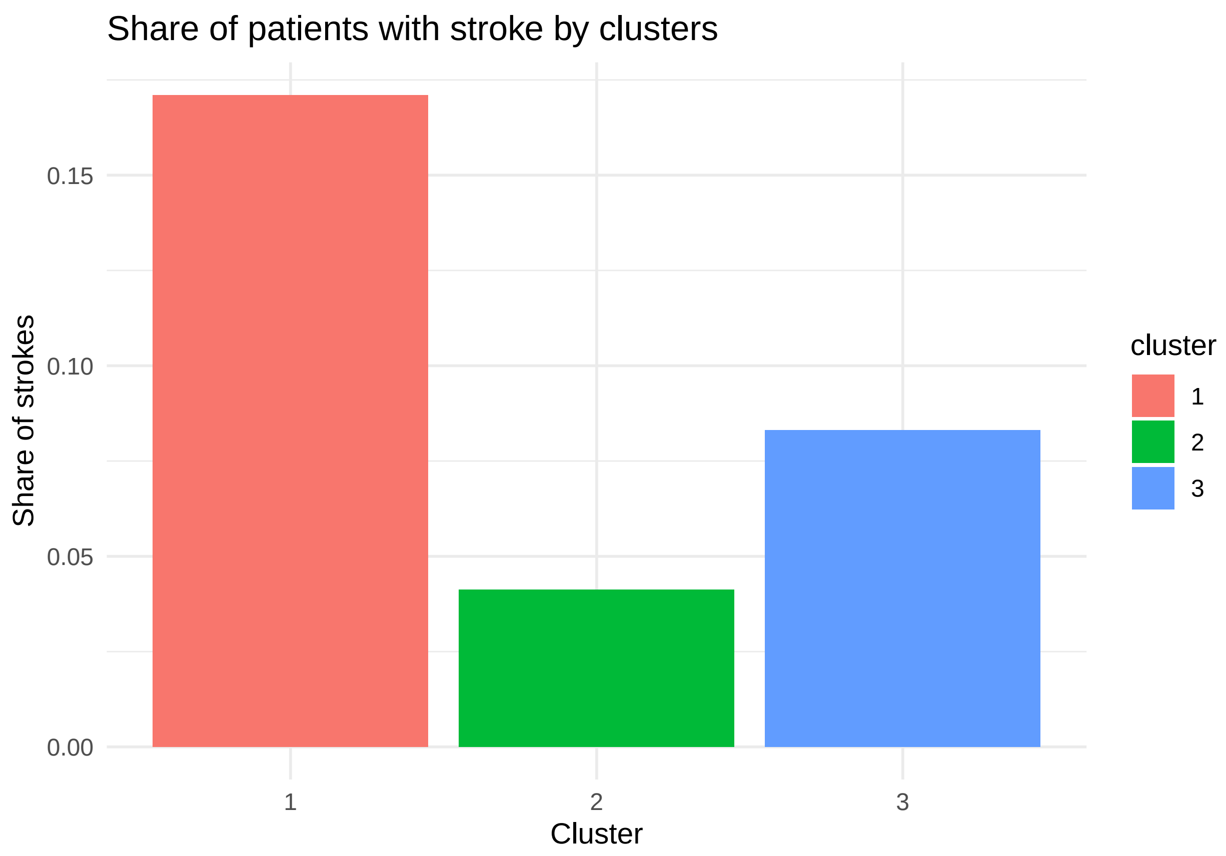
```
## [1] "Stroke shares by clusters"
```

```r
prop.table(table(clustered_data$cluster, clustered_data$stroke), margin = 1)
```

```
##
##              0          1
##   1 0.82894737 0.17105263
##   2 0.95868373 0.04131627
##   3 0.91694352 0.08305648
```

From the tables below it is evidenced that 1st class experience higher stroke rates (approximately 17%) in compassion to other clusters (4 and 8 %). Lets also visualize the share of strokes for each cluster.

```r
cluster_summary <- clustered_data %>%
  group_by(cluster) %>%
  summarise(stroke_rate = mean(as.numeric(as.character(stroke))))

ggplot(cluster_summary, aes(x = cluster, y = stroke_rate, fill = cluster)) +
  geom_col() +
  labs(title = "Share of patients with stroke by clusters",
       x = "Cluster",
       y = "Share of strokes") +
  theme_minimal()
```



As at can be seen on the bar chart, indeed cluster with the biggest share of strokes is first one, therefore, it is our risk group. Next, we will calculate the mean values for attributes for each cluster.

```r
library(dplyr)

#
cluster_summary <- clustered_data %>%
  group_by(cluster) %>%
  summarise(
    count = n(),
    avg_age = mean(age),
    avg_glucose = mean(avg_glucose_level),
    work_self = mean(work_self),
    hypertension_rate = mean(hypertension),
    heart_disease = mean(heart_disease)
  )

print(cluster_summary)
```

```
## # A tibble: 3 x 7
##   cluster count avg_age avg_glucose work_self hypertension_rate heart_disease
##   <fct>   <int>   <dbl>       <dbl>     <dbl>             <dbl>         <dbl>
## 1 1         228   0.805       0.375     0.268             0.259             1
## 2 2        2735   0.485       0.234     0                 0.101             0
## 3 3         602   0.688       0.268     1                 0.183             0
```

Before interpreting the results, it's important to point out the type of data we will see in the table. Since we used normalization for our numerical variables, the mean values will be normalized as well (Age, Glucose Level). For our binary variables (Work self, Hypertenstion, and Heart disease) we will get shares of patients with positive values (i.e. 1). Now, lets move to the results.

Table demonstrate means for each attribute of the cluster. We are interested in the first one. The average age for the risk group is approximately 68 years, since we used min-max normalization while preparing our data; average glucose level is approximately 136 mg/dL (again since we de-normalize our value), which corresponds to the 8 mmol/L (high sugar level in blood) according to some internet resources; approximately 27% of patients from the 1 cluster are self-employed; approximately 26% of the patients have hypertension; and 100% of patients have heart diseases.

Overall, it can be concluded the people that are close to 70 years, are self-employed, have heart disease and high average glucose level are in the risk group of experiencing stroke.

Conclusion

In our study project, we attempted to accomplish two tasks. First, we tried to develop a model that could predict stroke cases. Unfortunately, we were not able to accomplish this task, all models had poor performance. When comparing the models built, the best performance was observed for Random Forest with 5-fold Cross Validation and SMOTE. In the future, the constructed models should be improved by eliminating overfitting.

The second task, from our point of view, was fully completed. We successfully identified a risk group and calculated average metrics for this group.