

Applied Machine Learning and Predictive Modelling I: Modelling Stroke Data

Authors: Larissa Eisele, Fabian Lüthard, Yves Maillard

Module: Applied Machine Learning and Predictive Modelling I

Submitted on 10th of June, 2022

SUPERVISOR: MATTEO TANADINI AND DANIEL MEISTER

Table of Contents

1	Introduction	1
2	Importing Data	1
3	Data Cleaning	2
3.1	Summaries	3
3.2	Scatterplots	4
3.3	Boxplots	8
4	Methodology	14
5	Linear Models	15
6	Generalised Linear Model with family set to Poisson	27
7	Generalised Linear Model with family set to Binomial	29
8	Generalised Additive Model	44
9	Neural Network Yves	46
10	Support Vector Machine (Larissa)	46
11	Compare Models	53
12	OPTIONAL solve an optimisation problem	53
13	Conclusion	53

1 Introduction

Use case: We are a smart watch manufacturer working on a new feature for stroke prevention. According to the World Health Organization (WHO) stroke is the 2nd leading cause of death globally, responsible for approximately 11% of total deaths. We are going to analyze survey data that we plan to ask our users, complementing it with HR (Heart Rate) and CGM (Continuous Glucose Monitoring) data that our product already measures. We hope that our feature can prevent serious health issues and motivate our users to adopt healthier lifestyles.

We worked with a Stroke Prediction Data set from (<https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset?>).

In the following document, different calculation and models are used. The different models are intended to reflect both the teaching content from the course and the knowledge that the authors have gained during the learning process itself.

!! Hier evtl auch Schwierigkeit von Linear Models beschreiben??

2 Importing Data

The first step was to research the relevant data. The data was imported into R. For simplicity, not all of the code is included. However, all code can be found in the original R Markdown file.

```
stroke_data <- read_csv('./data/healthcare-dataset-stroke-data.csv')

## Rows: 5110 Columns: 12
## -- Column specification -----
## Delimiter: ","
## chr (6): gender, ever_married, work_type, Residence_type, bmi, smoking_status
## dbl (6): id, age, hypertension, heart_disease, avg_glucose_level, stroke
##
## i Use 'spec()' to retrieve the full column specification for this data.
## i Specify the column types or set 'show_col_types = FALSE' to quiet this message.

#stroke_data$bmi <- as.numeric(stroke_data$bmi)
#stroke_data$bmi[is.na(stroke_data$bmi)] <- mean(stroke_data$bmi, na.rm = TRUE)
stroke_data$age <- as.integer(stroke_data$age)
stroke_data$smoking_status <- as.factor(stroke_data$smoking_status)
stroke_data$work_type <- as.factor(stroke_data$work_type)
stroke_data$gender <- as.factor(stroke_data$gender)
stroke_data$ever_married <- as.factor(stroke_data$ever_married)
stroke_data$Residence_type <- as.factor(stroke_data$Residence_type)
stroke_data$stroke_num <- as.numeric(stroke_data$stroke)
stroke_data$stroke <- as.factor(stroke_data$stroke)

stroke_data

## # A tibble: 5,110 x 13
##       id gender   age hypertension heart_disease ever_married work_type
##   <dbl> <fct> <int>         <dbl>         <dbl> <fct>         <fct>
## 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
## 7 53882 Male 74 1 1 Yes Private
## 8 10434 Female 69 0 0 No Private
## 9 27419 Female 59 0 0 Yes Private
## 10 60491 Female 78 0 0 Yes Private
## # ... with 5,100 more rows, and 6 more variables: Residence_type <fct>,
## # avg_glucose_level <dbl>, bmi <chr>, smoking_status <fct>, stroke <fct>,
## # stroke_num <dbl>
```

3 Data Cleaning

The data has been prepared for an easier analysis and for fitting the models and calculations.

```
head(stroke_data)
```

```
## # A tibble: 6 x 13
##   id gender age hypertension heart_disease ever_married work_type
##   <dbl> <fct> <int>         <dbl>         <dbl> <fct>         <fct>
## 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
## # ... with 6 more variables: Residence_type <fct>, avg_glucose_level <dbl>,
## # bmi <chr>, smoking_status <fct>, stroke <fct>, stroke_num <dbl>
```

```
#rename parameter residence_type to lower case for stylistic purposes
stroke_data <- stroke_data %>% rename("residence_type" = "Residence_type")
```

```
#check for the dimension of the data set
dim(stroke_data)
```

```
## [1] 5110 13
```

The data set contains 201 missing values in “bmi”.

```
#check for missing values
colSums(is.na(stroke_data))
```

```
##           id           gender           age           hypertension
##           0             0             0             0
## heart_disease ever_married work_type residence_type
##           0             0             0             0
## avg_glucose_level      bmi smoking_status      stroke
##           0             0             0             0
## stroke_num
##           0
```

As the data set given is already quite small, we will replace the missing values with the mean “bmi” value.

```
#convert bmi as number and replace missing values with the mean value of bmi
stroke_data$bmi <- as.numeric(stroke_data$bmi)
```

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

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

!!! Ich weiss nicht ob wir hier genauer testen sollen ob das sinnvoll ist, da Matteo im R-Bootcamp recht klar gesagt hat, das der Umgang mit missing values immer sehr kritisch ist und man daher die daten besser löscht als ersetzt.

3.1 Summaries

```
#comparing gender with stroke
stroke_by_gender = table(stroke_data$gender, stroke_data$stroke)
names(dimnames(stroke_by_gender))<- c("Gender", "Stroke")
stroke_by_gender
```

```
##           Stroke
## Gender      0    1
## Female 2853  141
## Male   2007  108
## Other     1    0
```

```
#testing the effect of non smokers and smokers
count_by_smoke_status <- stroke_data %>%
  select(smoking_status, stroke) %>%
  group_by(smoking_status, stroke) %>%
  summarise( N = n())
```

```
## 'summarise()' has grouped output by 'smoking_status'. You can override using the
## '.groups' argument.
```

```
#testing the effect of work type
count_by_work_type <- stroke_data %>%
  select(work_type, stroke) %>%
  group_by(work_type, stroke) %>%
  summarise( N = n())
```

```
## 'summarise()' has grouped output by 'work_type'. You can override using the
## '.groups' argument.
```

```
# testing the effects of residence type
count_by_residence_type <- stroke_data %>%
  select(residence_type, stroke) %>%
  group_by(residence_type, stroke) %>%
  summarise( N = n())
```

```
## 'summarise()' has grouped output by 'residence_type'. You can override using the  
## '.groups' argument.
```

```
# testing the effects of gender  
count_by_gender <- stroke_data %>%  
  select(gender, stroke) %>%  
  group_by(gender, stroke) %>%  
  summarise( N = n())
```

```
## 'summarise()' has grouped output by 'gender'. You can override using the  
## '.groups' argument.
```

```
# testing the effects of hypertension  
count_by_hypertension <- stroke_data %>%  
  select(hypertension, stroke) %>%  
  group_by(hypertension, stroke) %>%  
  summarise( N = n())
```

```
## 'summarise()' has grouped output by 'hypertension'. You can override using the  
## '.groups' argument.
```

```
# testing the effects of heart disease  
count_by_heart_disease <- stroke_data %>%  
  select(heart_disease, stroke) %>%  
  group_by(heart_disease, stroke) %>%  
  summarise( N = n())
```

```
## 'summarise()' has grouped output by 'heart_disease'. You can override using the  
## '.groups' argument.
```

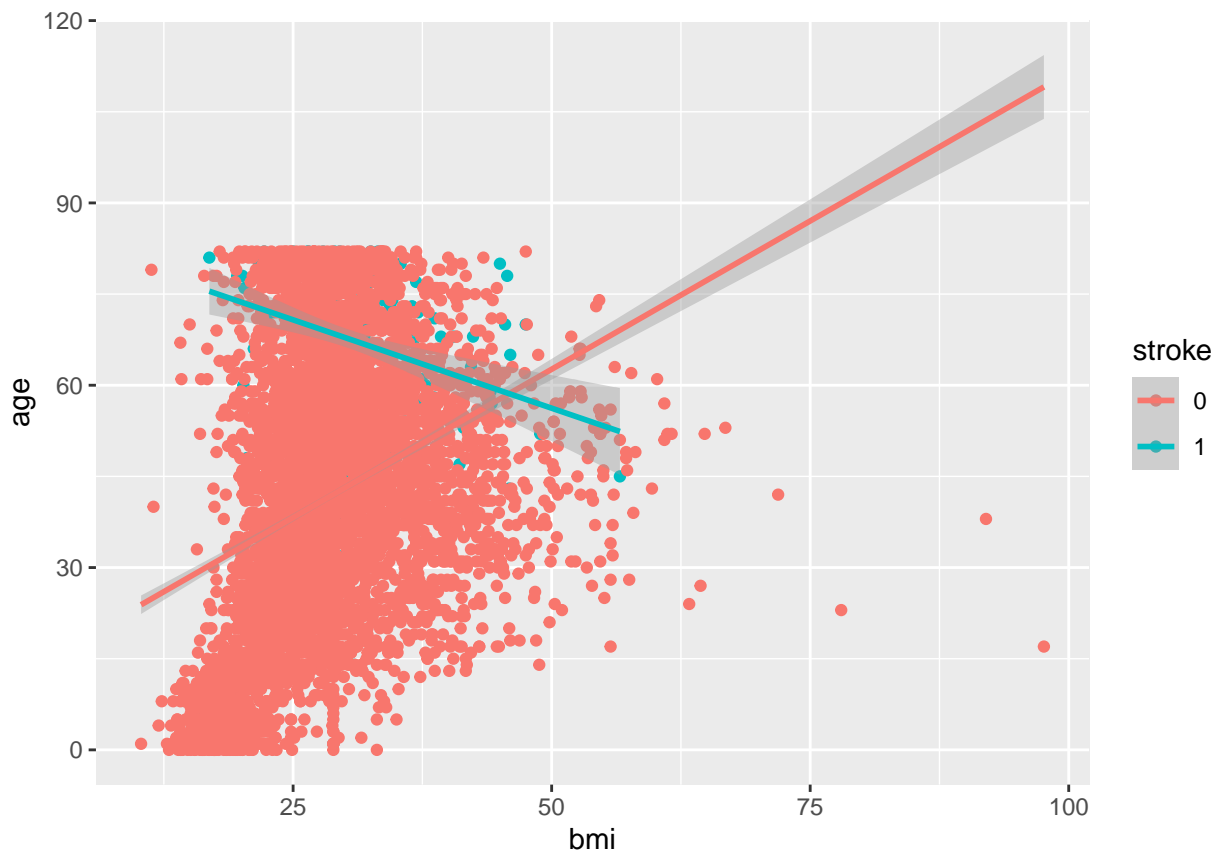
```
# testing the effects of marriage status  
count_by_marriage <- stroke_data %>%  
  select(ever_married, stroke) %>%  
  group_by(ever_married, stroke) %>%  
  summarise( N = n())
```

```
## 'summarise()' has grouped output by 'ever_married'. You can override using the  
## '.groups' argument.
```

3.2 Scatterplots

```
stroke_data %>%  
  ggplot(mapping = aes(x = bmi, y = age, color = stroke)) +  
  geom_point() +  
  geom_smooth(method = 'lm')
```

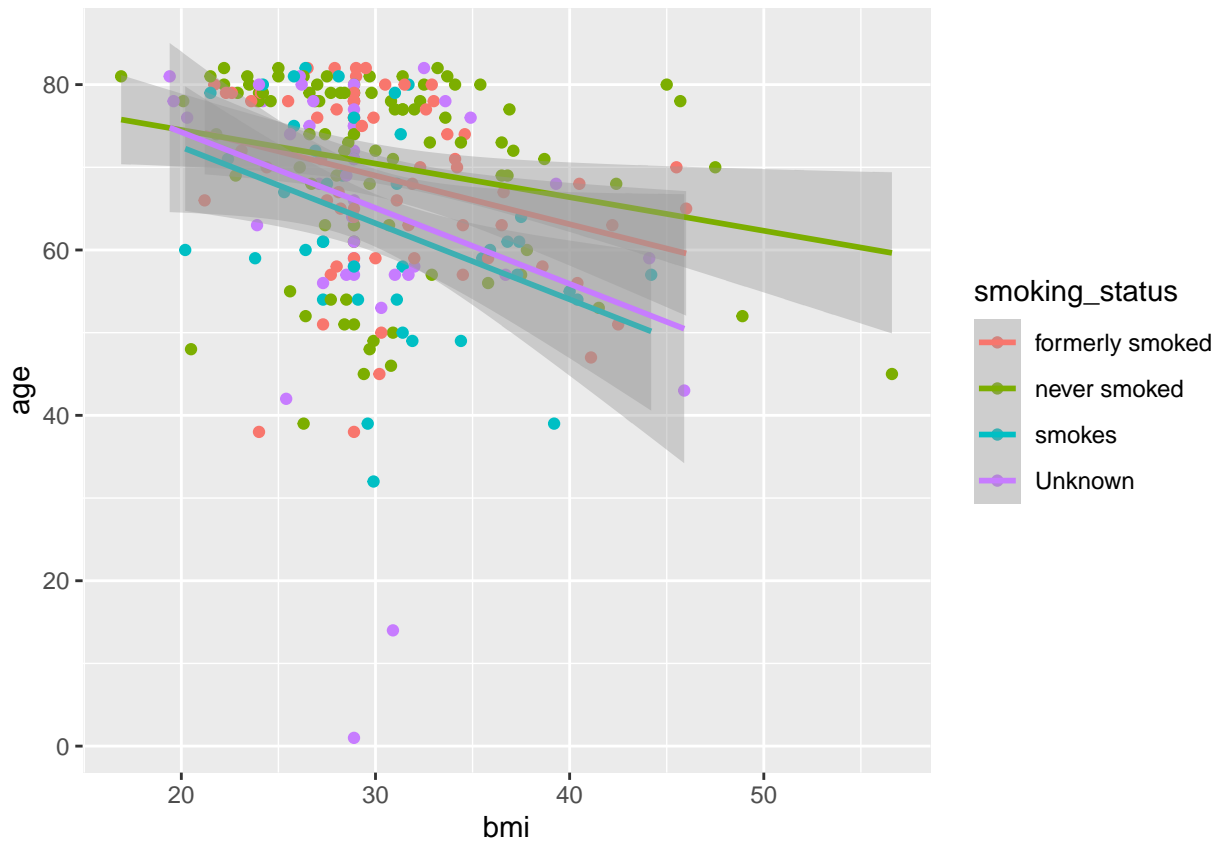
```
## 'geom_smooth()' using formula 'y ~ x'
```



```
stroke_data %>%  
  filter(stroke == 1) %>%  
  ggplot(mapping = aes(x = bmi, y = age, color = smoking_status)) +  
  geom_point(method = 'lm') +  
  geom_smooth(method = 'lm')
```

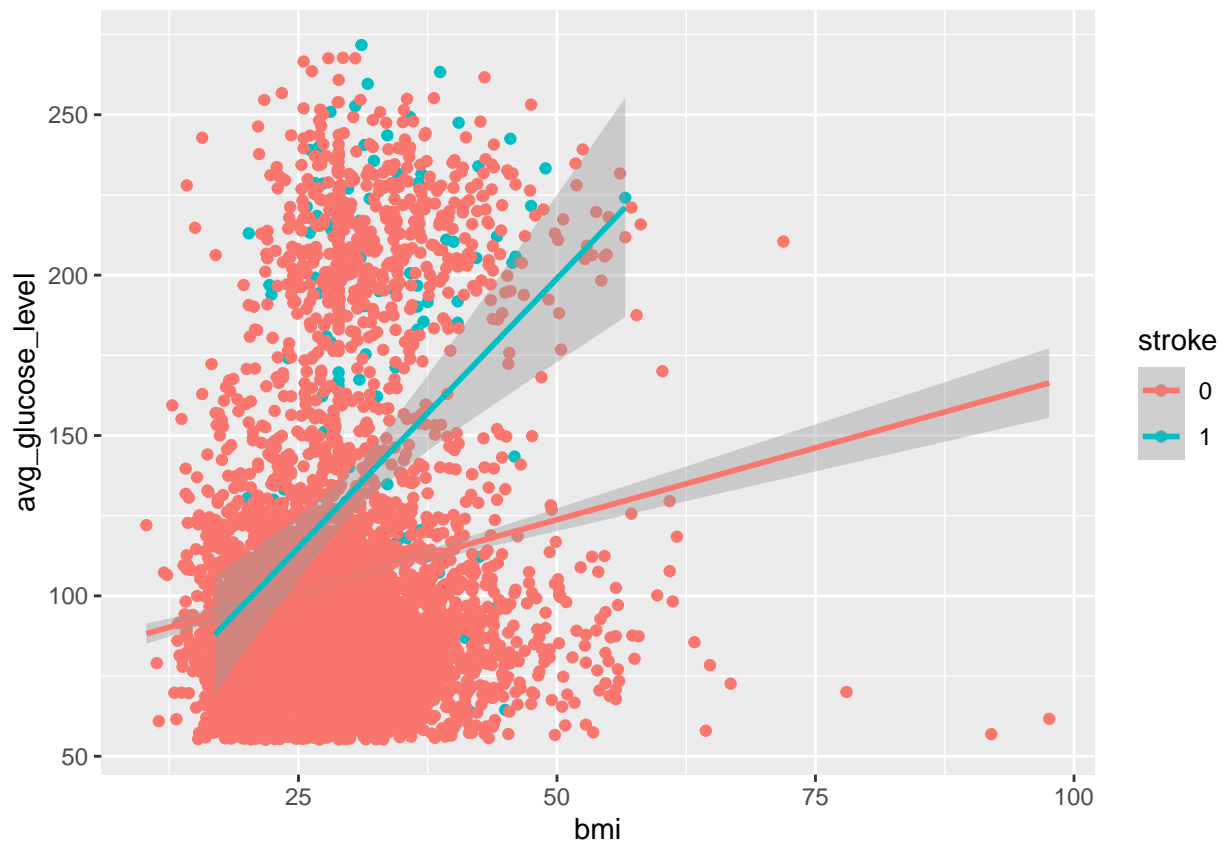
```
## Warning: Ignoring unknown parameters: method
```

```
## 'geom_smooth()' using formula 'y ~ x'
```



```
stroke_data %>%
  ggplot(mapping = aes(x = bmi, y = avg_glucose_level, color = stroke)) +
  geom_point() +
  geom_smooth(method = 'lm')
```

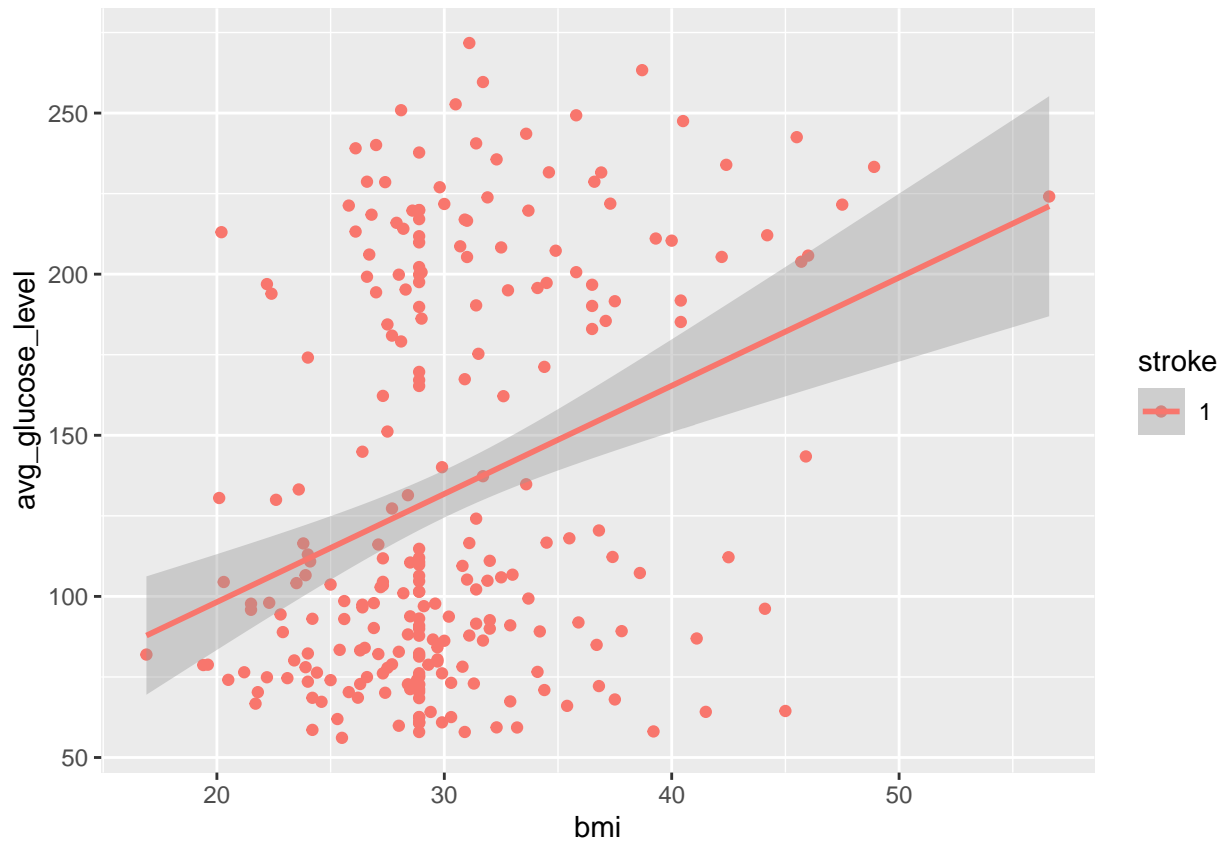
```
## 'geom_smooth()' using formula 'y ~ x'
```

```
stroke_data %>%  
  filter(stroke == 1) %>%  
  ggplot(mapping = aes(x = bmi, y = avg_glucose_level, color = stroke)) +  
  geom_point(method = 'lm') +  
  geom_smooth(method = 'lm')
```

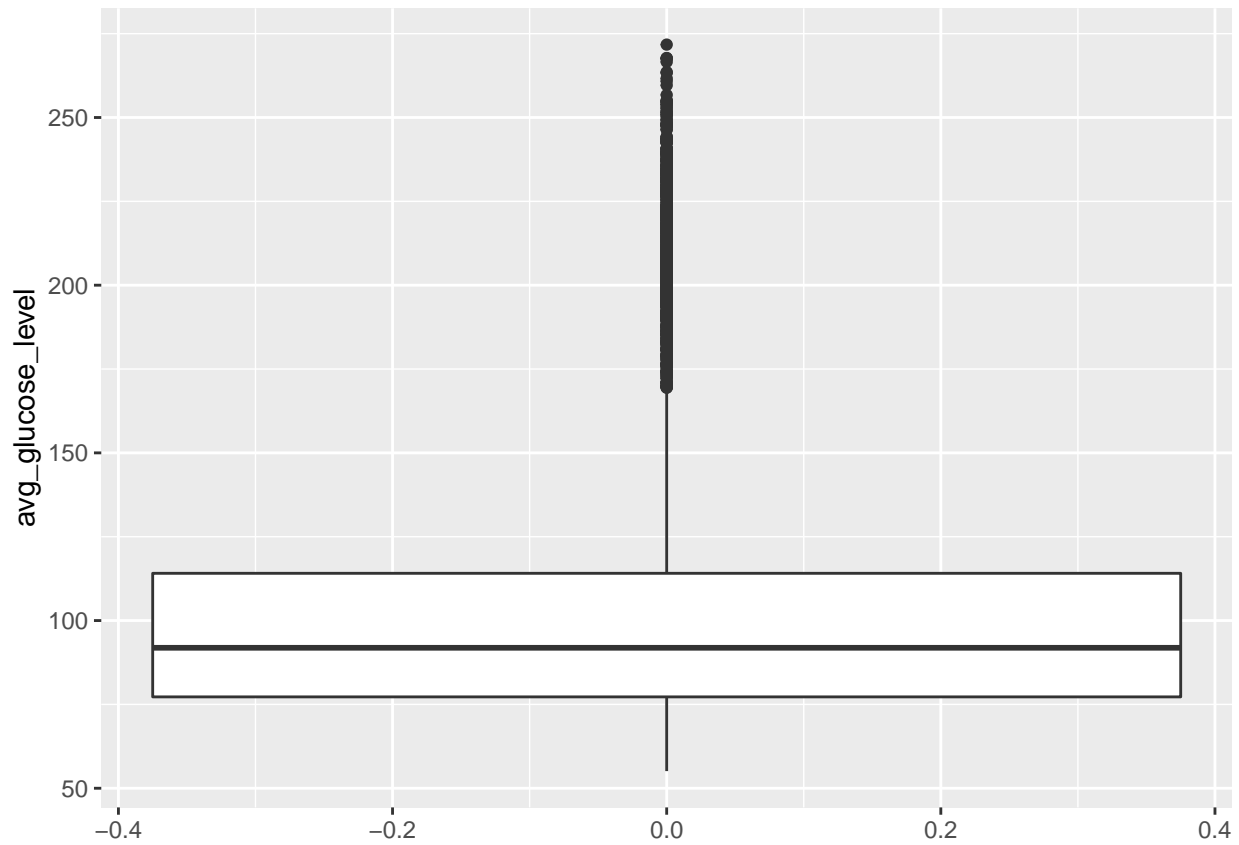
```
## Warning: Ignoring unknown parameters: method
```

```
## 'geom_smooth()' using formula 'y ~ x'
```

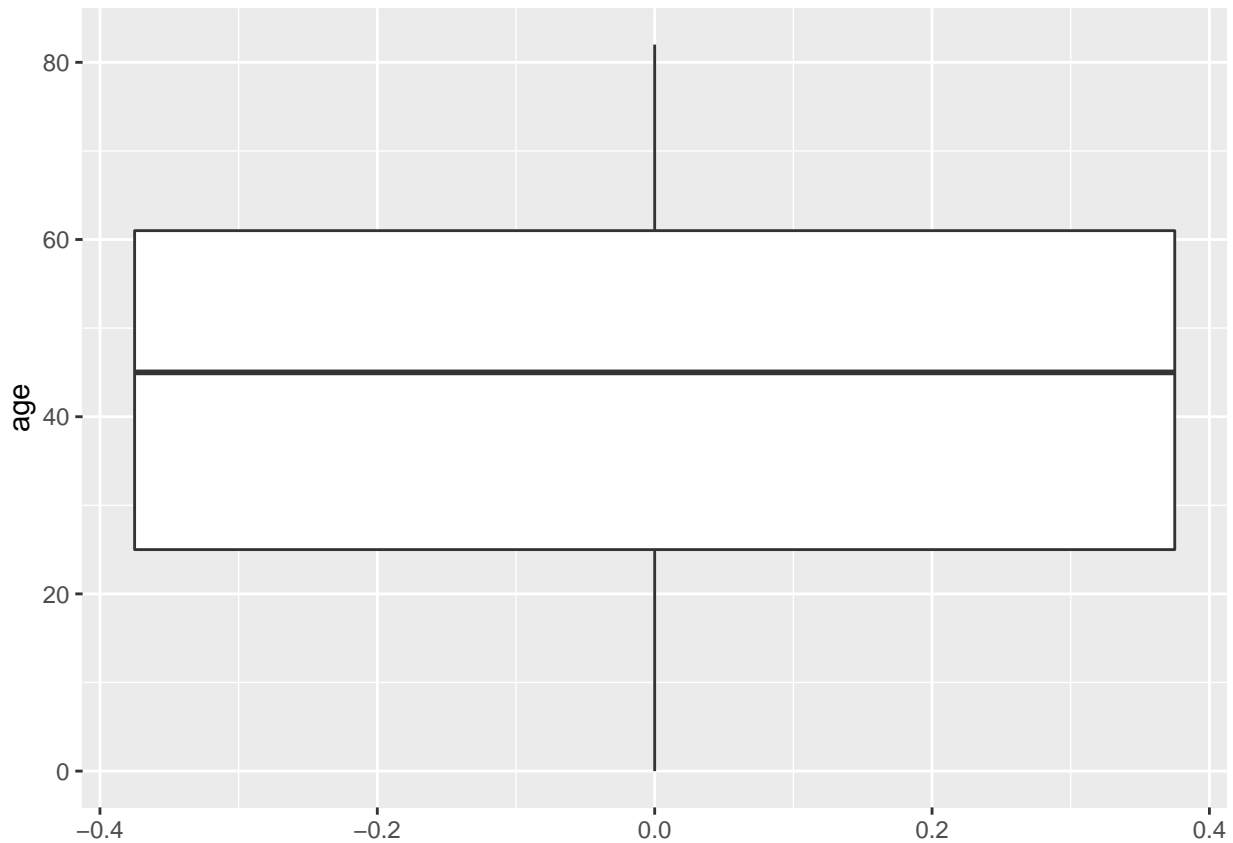


3.3 Boxplots

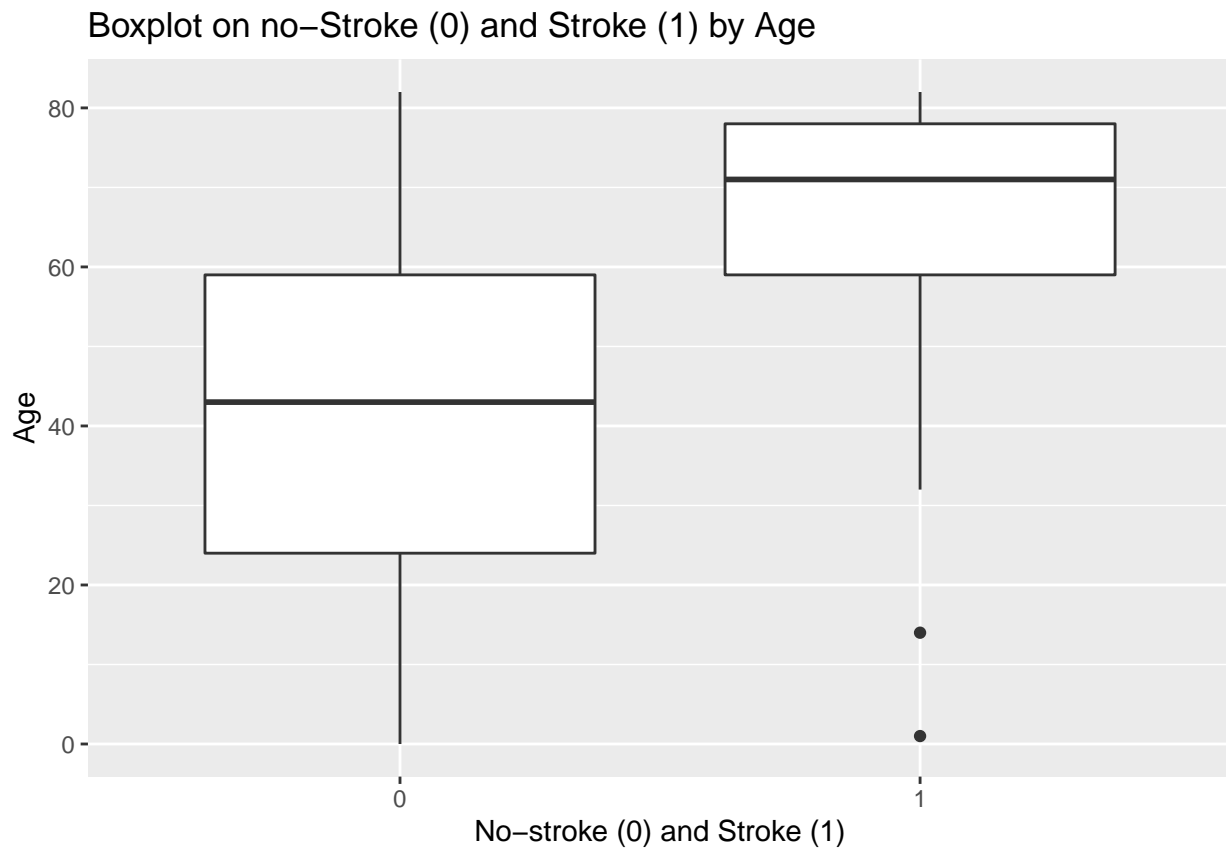
```
#Boxplot on avg. glucose_level  
ggplot(stroke_data, aes(y = avg_glucose_level)) +  
  geom_boxplot()
```



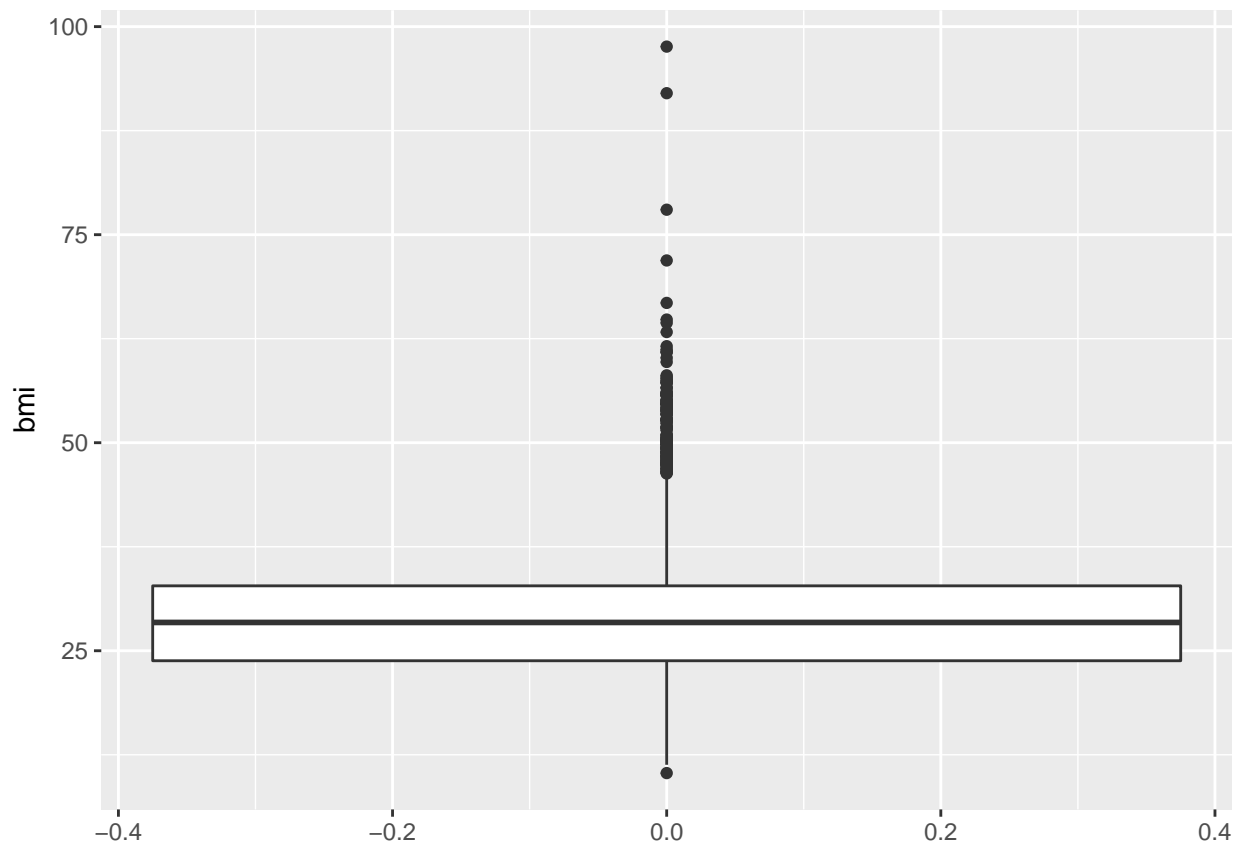
```
#Boxplot on Age  
ggplot(stroke_data, aes(y = age)) +  
  geom_boxplot()
```



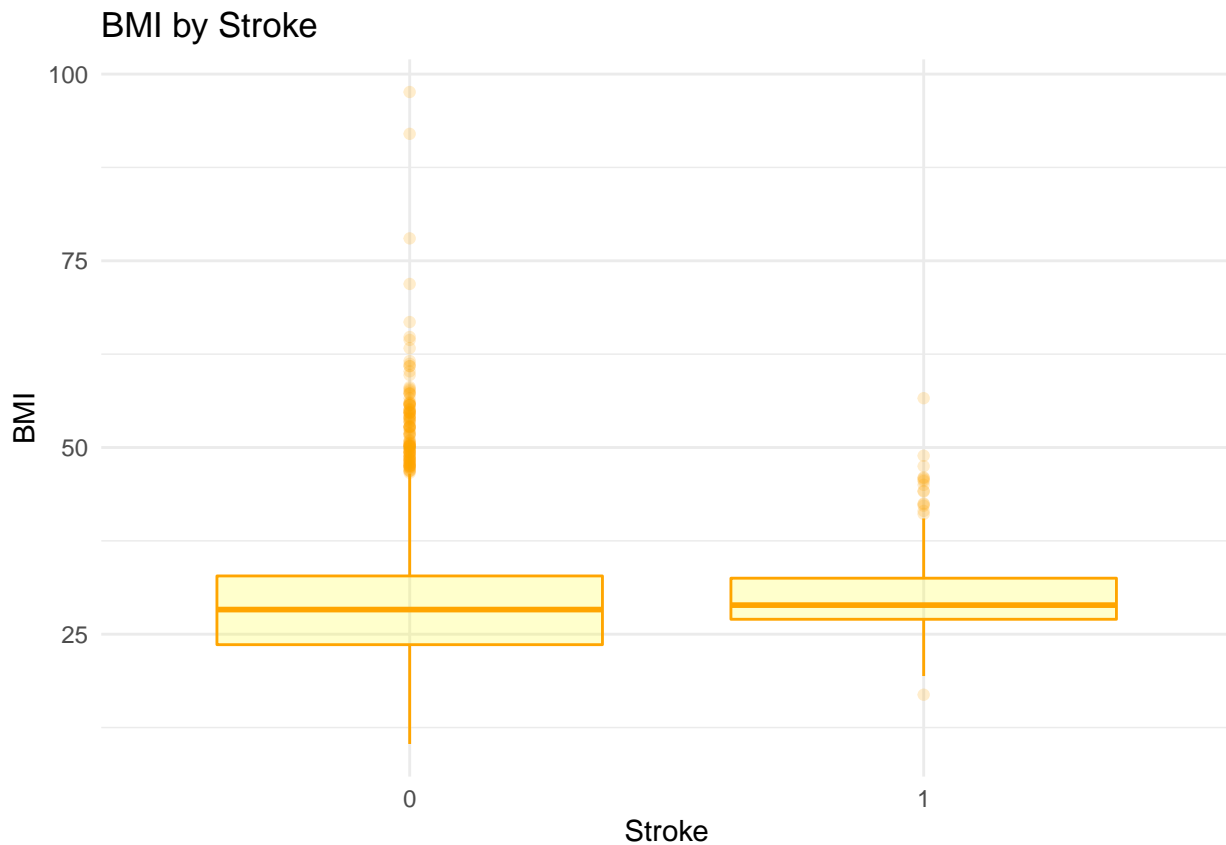
```
stroke_by_age <- stroke_data %>%  
# dplyr::filter(stroke == 1) %>%  
ggplot(aes(x = stroke,  
           y = age)) +  
geom_boxplot() +  
labs(title = "Boxplot on no-Stroke (0) and Stroke (1) by Age",  
      x = "No-stroke (0) and Stroke (1)",  
      y = "Age")  
color = "green"  
  
stroke_by_age
```



```
#Boxplot on BMI  
ggplot(stroke_data, aes(y = bmi)) +  
  geom_boxplot()
```

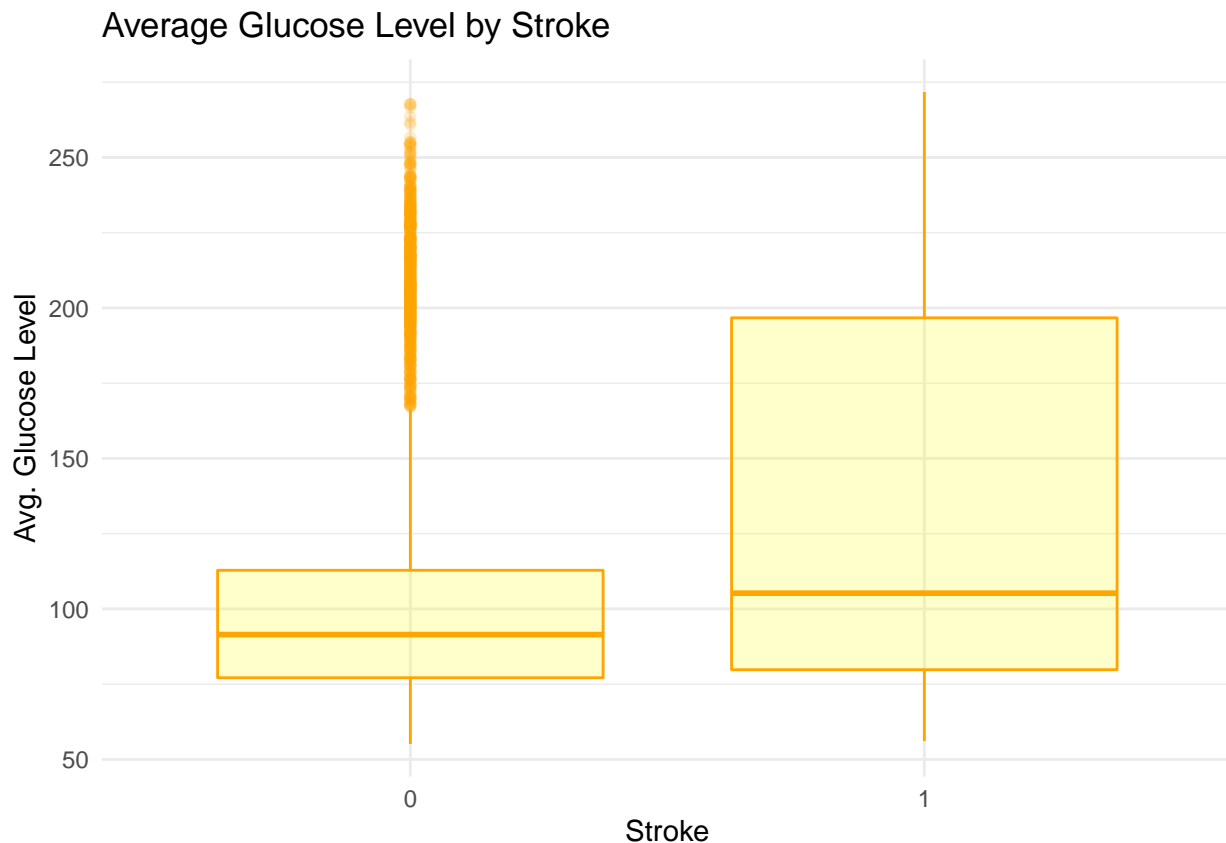


```
stroke_by_bmi <- stroke_data %>%  
# dplyr::filter(stroke == 1) %>%  
ggplot(aes(x = stroke,  
           y = bmi)) +  
  geom_boxplot(color="orange", fill="yellow", alpha=0.2) +  
  ggtitle("BMI by Stroke") +  
  xlab("Stroke") + ylab("BMI") +  
  theme_minimal() + theme(axis.text.x = element_text(angle = 0))  
stroke_by_bmi
```



```
#boxplot by avg_glucose_level and stroke
stroke_by_avg_glucose_level <- stroke_data %>%
# dplyr::filter(stroke == 1) %>%
ggplot(aes(x = stroke,
            y = avg_glucose_level)) +
  geom_boxplot(color="orange", fill="yellow", alpha=0.2) +
  ggtitle("Average Glucose Level by Stroke") +
  xlab("Stroke") + ylab("Avg. Glucose Level") +
  theme_minimal() + theme(axis.text.x = element_text(angle = 0))

stroke_by_avg_glucose_level
```



4 Methodology

```
set.seed(7406)
n=dim(stroke_data[1]) # number of observations in dataset
n_train=0.70*n # training set is 70%
flag = sort(sample(1:n, size=n_train, replace=FALSE))
```

```
## Warning in 1:n: numerical expression has 2 elements: only the first used
```

```
# Use df (all data points without ID column) df_train, and df_test
# Gender, hypertension, heart disease, ever married, work type, residence type, smoking status, and str
# This should allow for the best modeling options possible for our methods.
df_train = stroke_data[flag,]
df_test = stroke_data[-flag,]
```

```
head(df_test)
```

```
## # A tibble: 6 x 13
##   id gender  age hypertension heart_disease ever_married work_type
##   <dbl> <fct> <int>         <dbl>         <dbl> <fct>         <fct>
## 1 51676 Female  61           0           0 Yes          Self-employed
## 2 10434 Female  69           0           0 No           Private
## 3 60491 Female  78           0           0 Yes          Private
```



```
## 4 12095 Female    61          0          1 Yes      Govt_job
## 5 58202 Female    50          1          0 Yes      Self-employed
## 6 56112 Male     64          0          1 Yes      Private
## # ... with 6 more variables: residence_type <fct>, avg_glucose_level <dbl>,
## #   bmi <dbl>, smoking_status <fct>, stroke <fct>, stroke_num <dbl>
```

```
head(df_train)
```

```
## # A tibble: 6 x 13
##   id gender  age hypertension heart_disease ever_married work_type
##   <dbl> <fct> <int>         <dbl>         <dbl> <fct>         <fct>
## 1  9046 Male   67           0           1 Yes      Private
## 2 31112 Male   80           0           1 Yes      Private
## 3 60182 Female 49           0           0 Yes      Private
## 4  1665 Female 79           1           0 Yes      Self-employed
## 5 56669 Male   81           0           0 Yes      Private
## 6 53882 Male   74           1           1 Yes      Private
## # ... with 6 more variables: residence_type <fct>, avg_glucose_level <dbl>,
## #   bmi <dbl>, smoking_status <fct>, stroke <fct>, stroke_num <dbl>
```

5 Linear Models

```
set.seed(7406)
n=dim(stroke_data[1]) # number of observations in dataset
n_train=0.70*n # training set is 70%
flag = sort(sample(1:n, size=n_train, replace=FALSE))
```

```
## Warning in 1:n: numerical expression has 2 elements: only the first used
```

```
# Use df (all data points without ID column) df_train_linear, and df_test_linear
# Gender, hypertension, heart disease, ever married, work type, residence type, smoking status, and stroke
# This should allow for the best modeling options possible for our methods.
df_train_linear = stroke_data[flag,]
df_test_linear = stroke_data[-flag,]
```

```
head(df_test_linear)
```

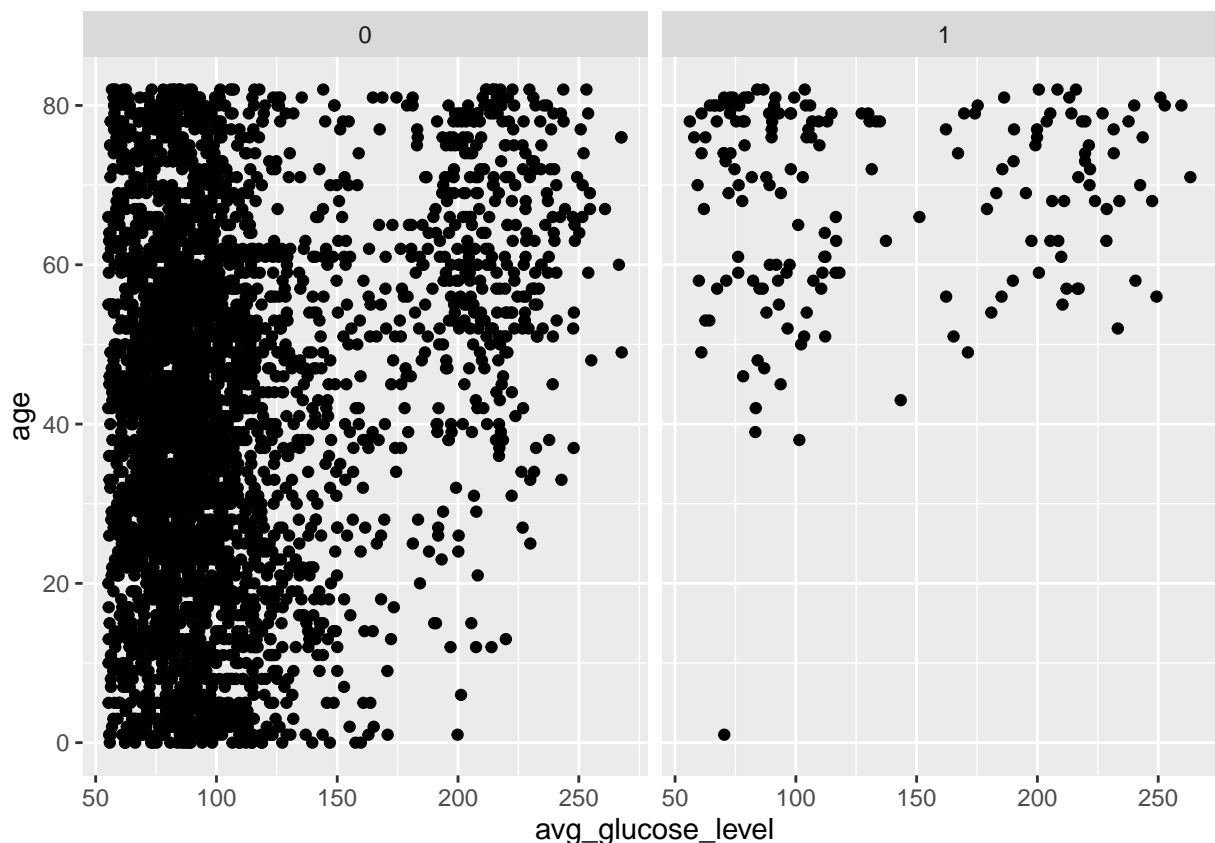
```
## # A tibble: 6 x 13
##   id gender  age hypertension heart_disease ever_married work_type
##   <dbl> <fct> <int>         <dbl>         <dbl> <fct>         <fct>
## 1 51676 Female  61           0           0 Yes      Self-employed
## 2 10434 Female  69           0           0 No       Private
## 3 60491 Female  78           0           0 Yes      Private
## 4 12095 Female  61           0           1 Yes      Govt_job
## 5 58202 Female  50           1           0 Yes      Self-employed
## 6 56112 Male   64           0           1 Yes      Private
## # ... with 6 more variables: residence_type <fct>, avg_glucose_level <dbl>,
## #   bmi <dbl>, smoking_status <fct>, stroke <fct>, stroke_num <dbl>
```

```
head(df_train_linear)
```

```
## # A tibble: 6 x 13
##   id gender  age hypertension heart_disease ever_married work_type
##   <dbl> <fct> <int>         <dbl>         <dbl> <fct>         <fct>
## 1  9046 Male    67             0             1 Yes         Private
## 2 31112 Male    80             0             1 Yes         Private
## 3 60182 Female  49             0             0 Yes         Private
## 4  1665 Female  79             1             0 Yes         Self-employed
## 5 56669 Male    81             0             0 Yes         Private
## 6 53882 Male    74             1             1 Yes         Private
## # ... with 6 more variables: residence_type <fct>, avg_glucose_level <dbl>,
## #   bmi <dbl>, smoking_status <fct>, stroke <fct>, stroke_num <dbl>
```

Our aim is to predict strokes, and the stroke variable is always our response variable of interest. However, simple or multiple linear regression is not the tool of choice, as stroke is a binary / categorical response variable. Thus in the following we will nonetheless fit a linear regression model to our data with stroke as the response variable (and another example to show that we've understood the concept). We know of the following limitations for fitting a linear regression model to a binary response variable: - - -

```
# qplot -> age/glucose level per stroke
library(ggplot2)
qplot(y = age, x = avg_glucose_level,
      data = df_train_linear,
      facets = ~ stroke)
```



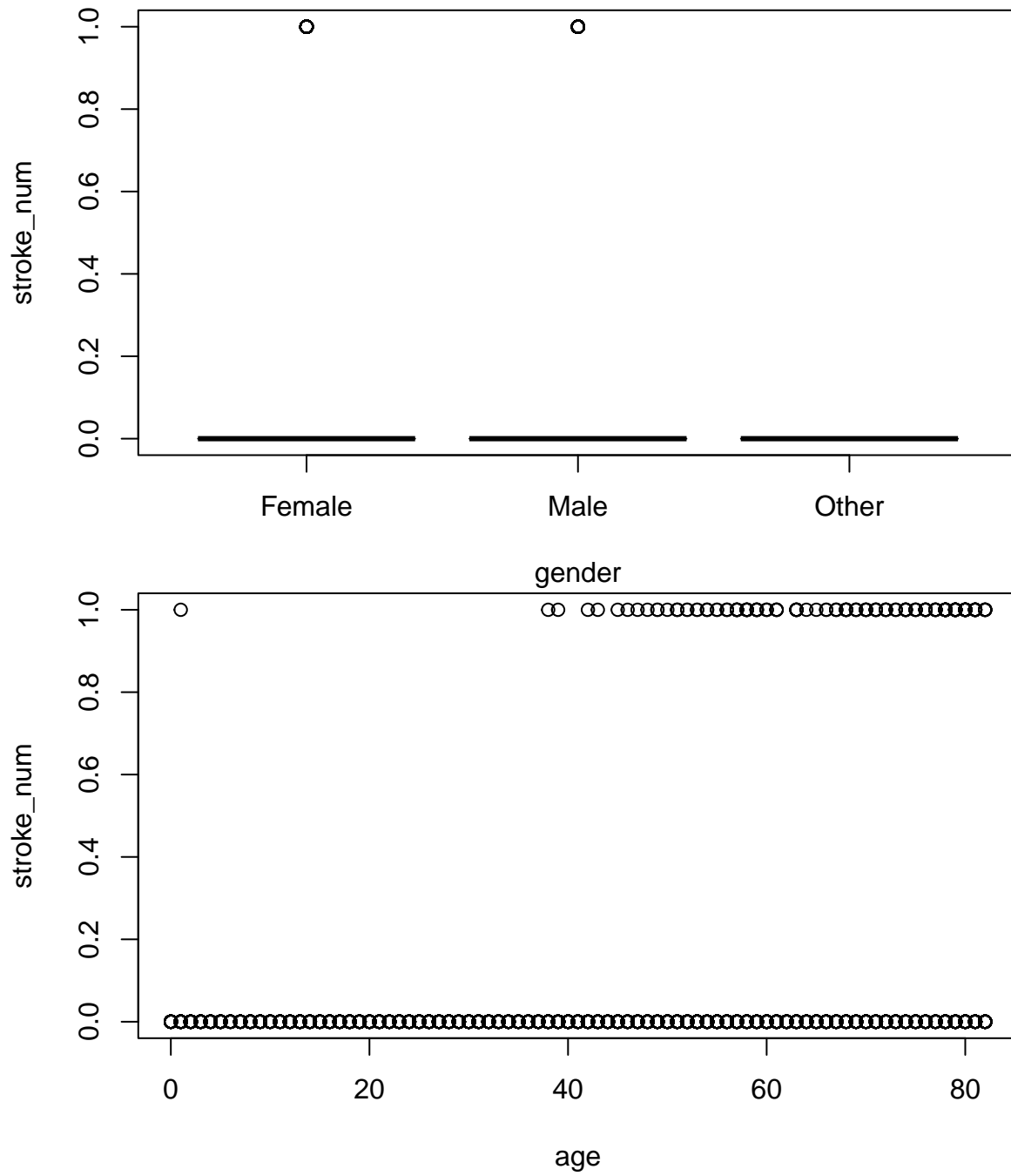
```

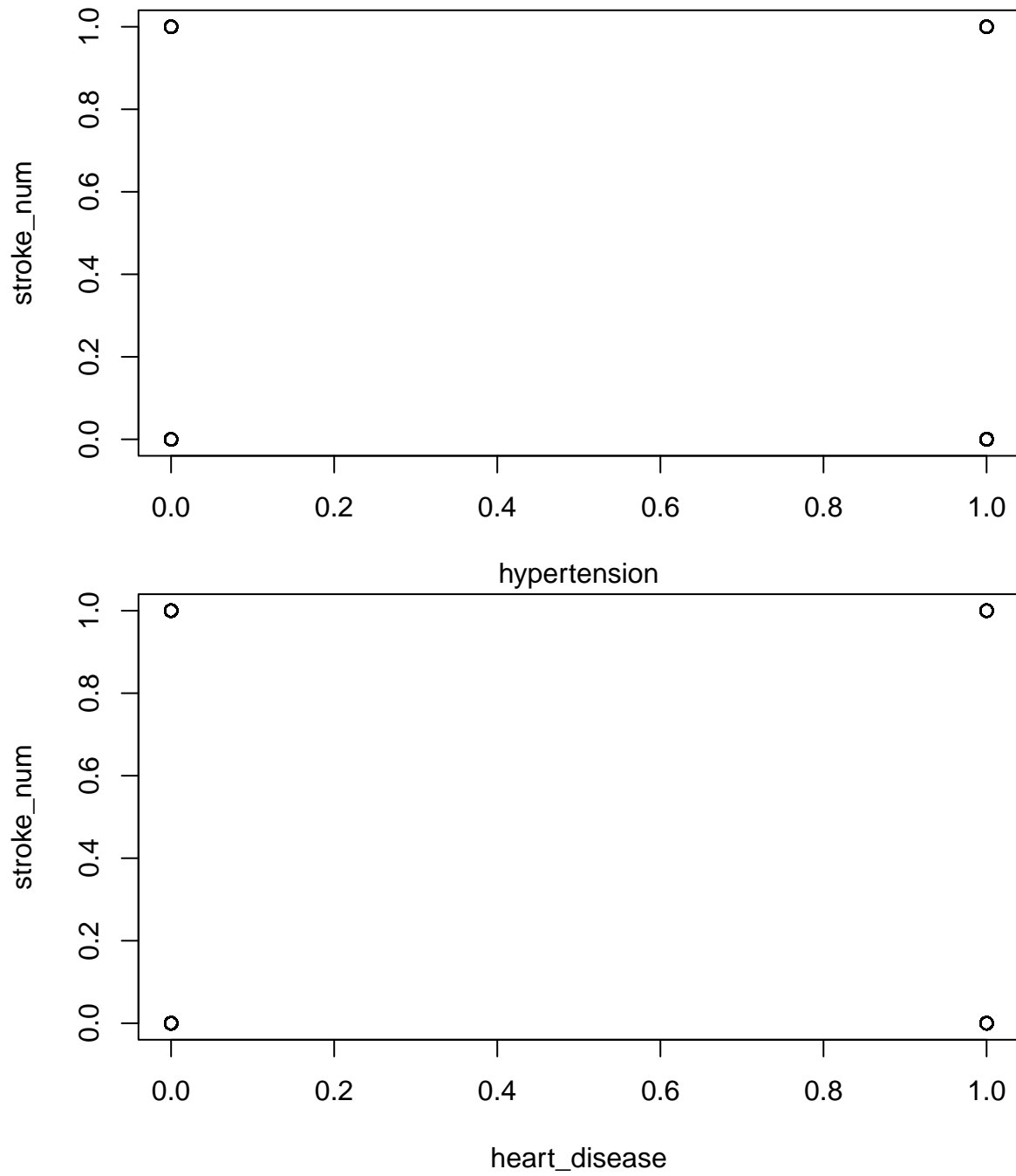
# fitting models for simple regression model
lm.stroke <- lm(stroke_num ~ gender + age + hypertension + heart_disease + ever_married + work_type + r
summary(lm.stroke)

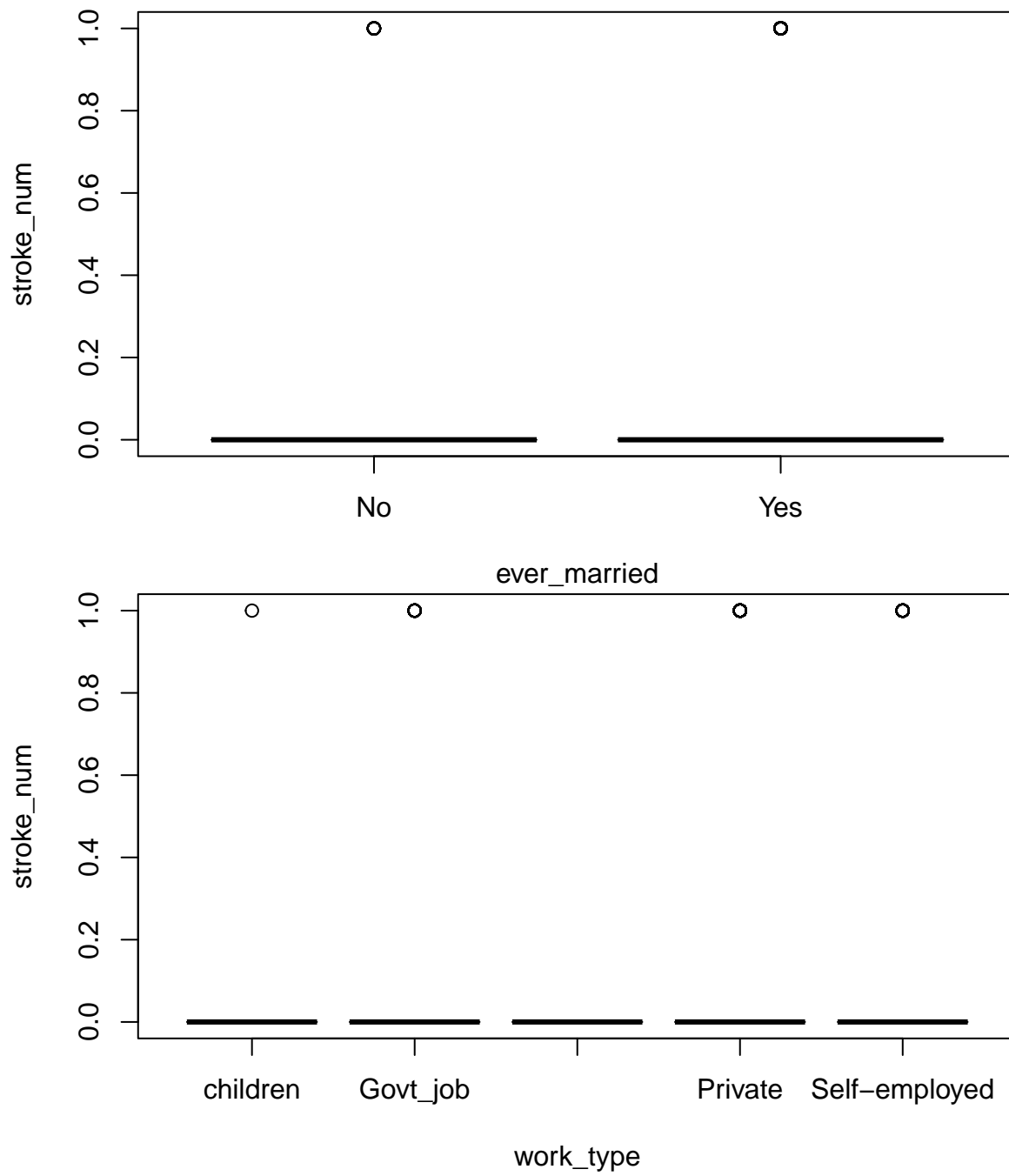
##
## Call:
## lm(formula = stroke_num ~ gender + age + hypertension + heart_disease +
##     ever_married + work_type + residence_type + avg_glucose_level +
##     bmi + smoking_status, data = df_train_linear)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.33328 -0.08333 -0.02421  0.00834  1.03042
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -3.812e-02  2.009e-02  -1.898  0.057814 .
## genderMale      4.560e-03  7.220e-03   0.632  0.527713
## genderOther    -2.698e-02  2.099e-01  -0.129  0.897733
## age            3.302e-03  2.617e-04  12.620 < 2e-16 ***
## hypertension   3.704e-02  1.210e-02   3.062  0.002218 **
## heart_disease   7.191e-02  1.598e-02   4.500  7.01e-06 ***
## ever_marriedYes -3.888e-02  1.045e-02  -3.721  0.000202 ***
## work_typeGovt_job -6.863e-02  1.854e-02  -3.701  0.000218 ***
## work_typeNever_worked -2.941e-02  5.372e-02  -0.548  0.584045
## work_typePrivate -5.075e-02  1.558e-02  -3.256  0.001139 **
## work_typeSelf-employed -8.413e-02  1.892e-02  -4.447  8.99e-06 ***
## residence_typeUrban  6.466e-04  7.016e-03   0.092  0.926575
## avg_glucose_level  2.898e-04  8.024e-05   3.612  0.000308 ***
## bmi           -5.155e-04  5.197e-04  -0.992  0.321284
## smoking_statusnever smoked -7.336e-03  1.040e-02  -0.705  0.480548
## smoking_statussmokes -5.882e-03  1.240e-02  -0.474  0.635307
## smoking_statusUnknown -1.745e-03  1.163e-02  -0.150  0.880725
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2095 on 3560 degrees of freedom
## Multiple R-squared:  0.09588,    Adjusted R-squared:  0.09181
## F-statistic: 23.59 on 16 and 3560 DF,  p-value: < 2.2e-16

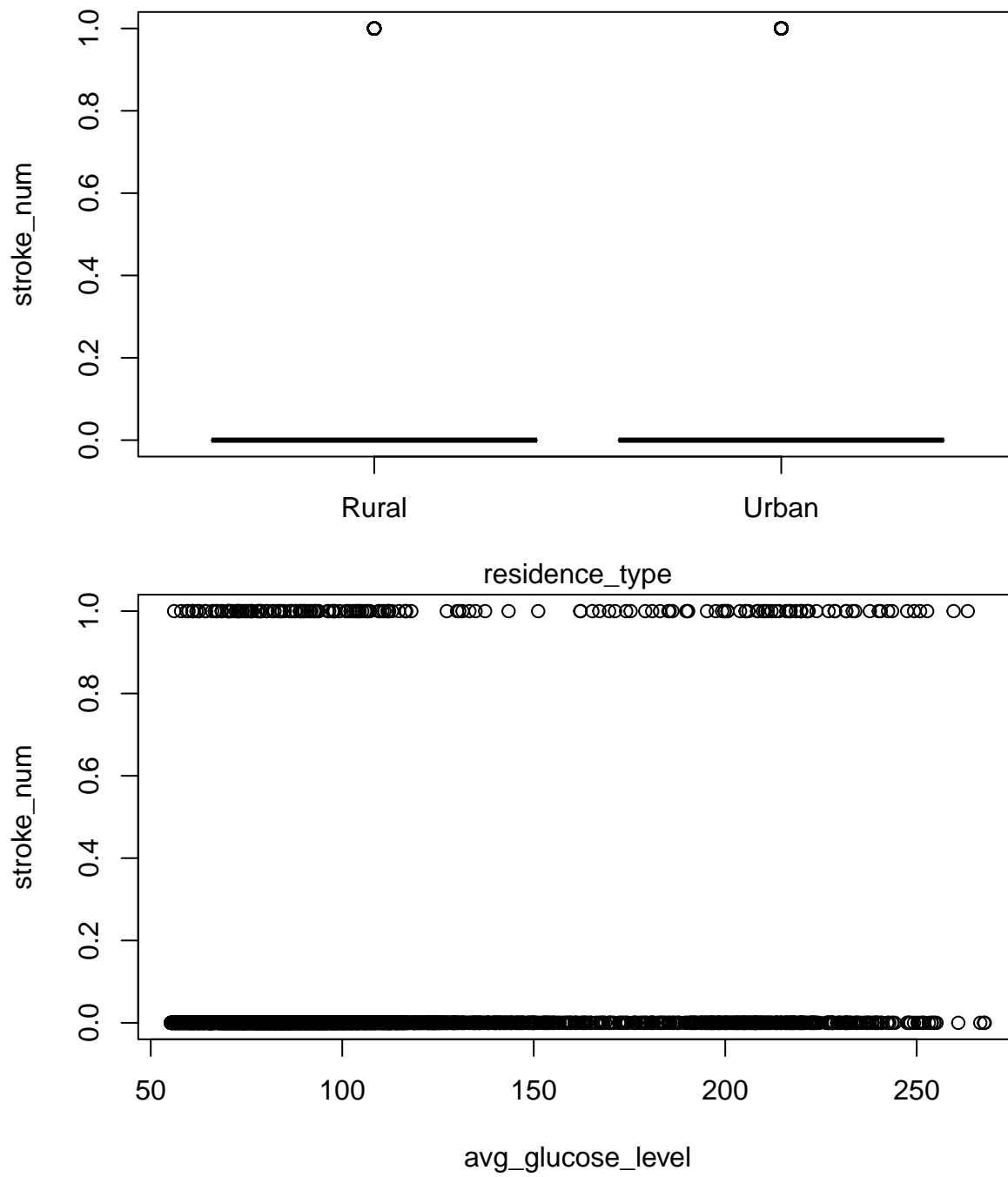
plot(stroke_num ~ gender + age + hypertension + heart_disease + ever_married + work_type + residence_ty

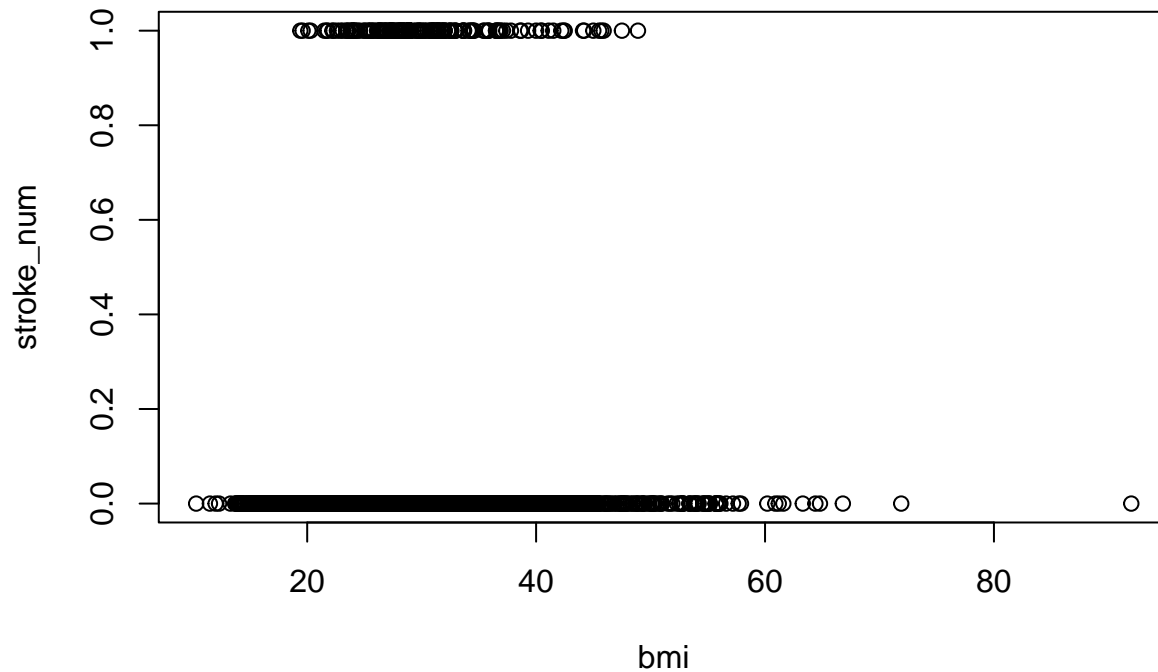
```





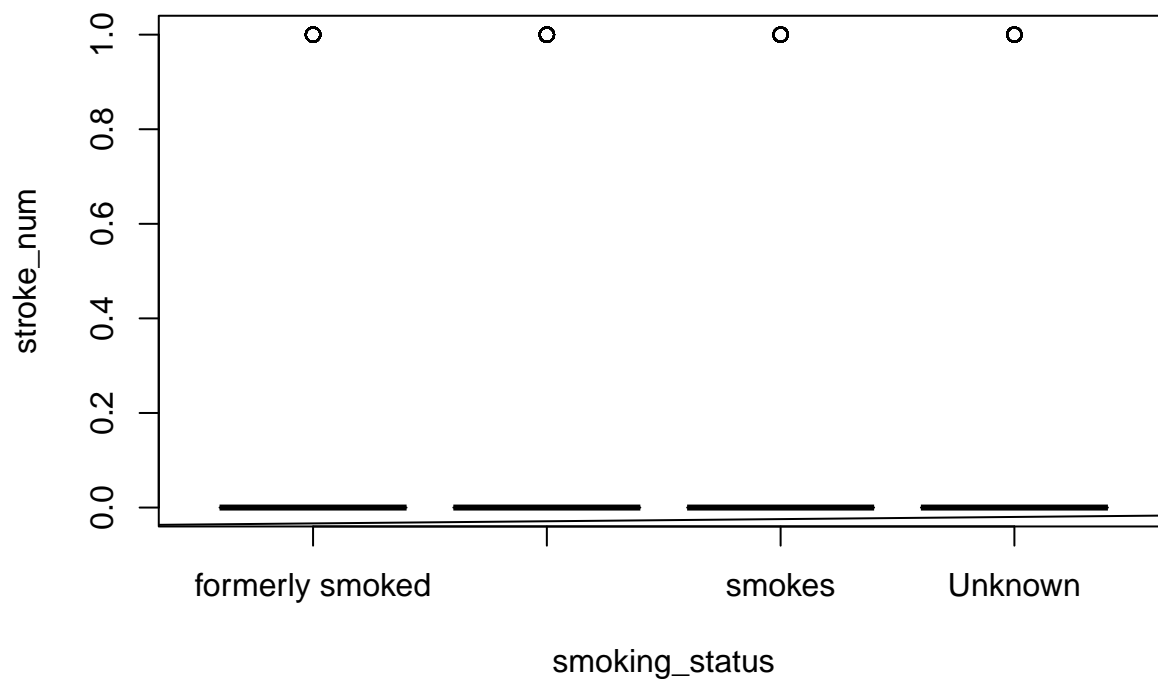






```
abline(lm.stroke)
```

```
## Warning in abline(lm.stroke): only using the first two of 17 regression
## coefficients
```



```
# fitting models for simple regression model
lm.stroke.smoking <- lm(stroke_num ~ smoking_status , data = df_train_linear)
summary(lm.stroke.smoking)
```

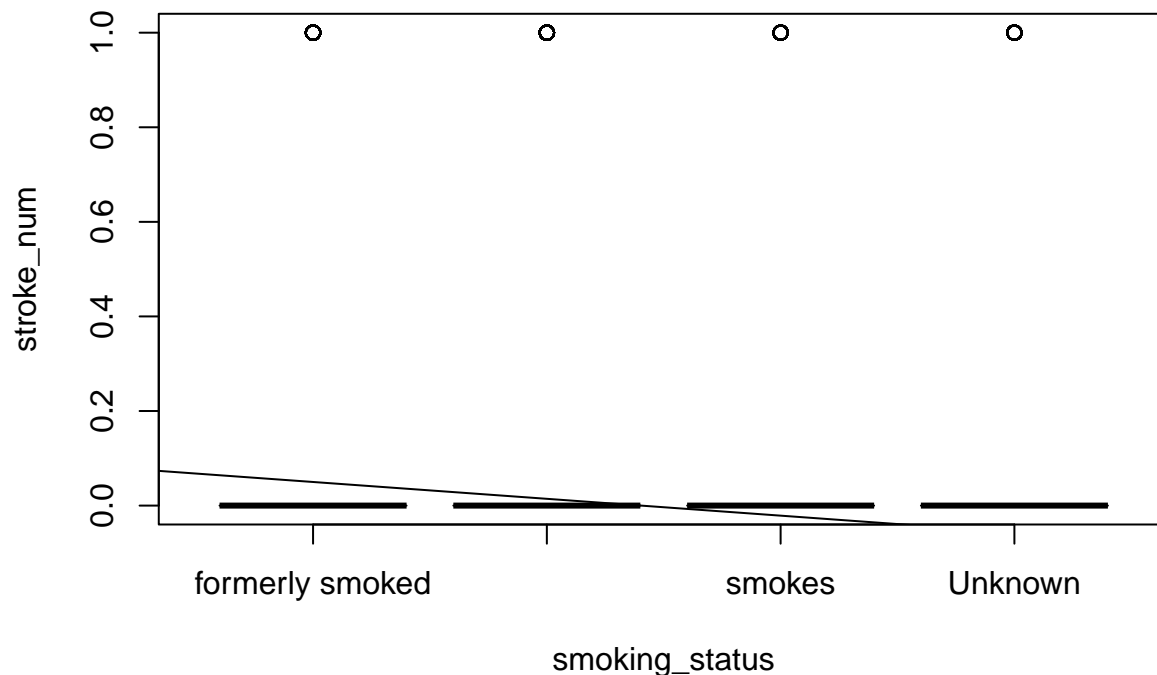
```
##
```



```
## Call:
## lm(formula = stroke_num ~ smoking_status, data = df_train_linear)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.08548 -0.05244 -0.04989 -0.03145  0.96855
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)      0.085484   0.008801   9.713 < 2e-16 ***
## smoking_statusnever smoked -0.035597   0.010666  -3.338 0.000854 ***
## smoking_statussmokes    -0.033043   0.012818  -2.578 0.009981 **
## smoking_statusUnknown   -0.054032   0.011040  -4.894 1.03e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2191 on 3573 degrees of freedom
## Multiple R-squared:  0.006675, Adjusted R-squared:  0.005841
## F-statistic: 8.004 on 3 and 3573 DF, p-value: 2.579e-05
```

```
plot(stroke_num ~ smoking_status, data = df_train_linear)
abline(lm.stroke.smoking)
```

```
## Warning in abline(lm.stroke.smoking): only using the first two of 4 regression
## coefficients
```

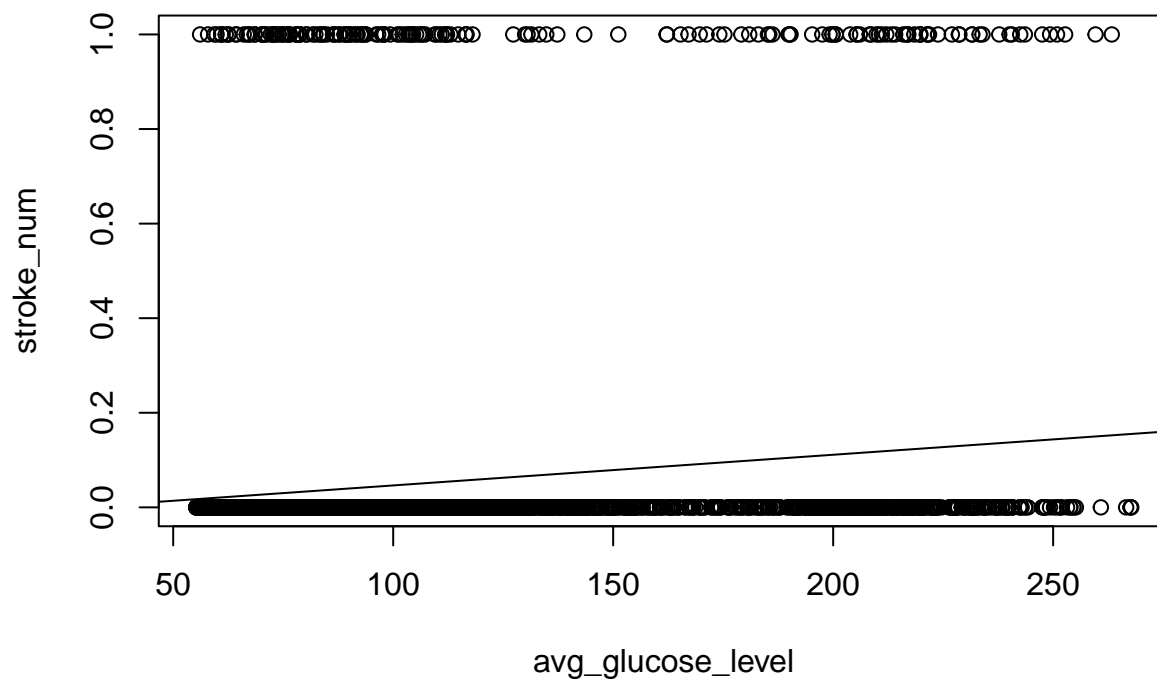


```
# fitting models for simple regression model
lm.stroke.glucose <- lm(stroke_num ~ avg_glucose_level , data = df_train_linear)
summary(lm.stroke.smoking)
```

```
##
```

```
## Call:
## lm(formula = stroke_num ~ smoking_status, data = df_train_linear)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.08548 -0.05244 -0.04989 -0.03145  0.96855
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)      0.085484   0.008801   9.713 < 2e-16 ***
## smoking_statusnever smoked -0.035597   0.010666  -3.338 0.000854 ***
## smoking_statussmokes    -0.033043   0.012818  -2.578 0.009981 **
## smoking_statusUnknown    -0.054032   0.011040  -4.894 1.03e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2191 on 3573 degrees of freedom
## Multiple R-squared:  0.006675, Adjusted R-squared:  0.005841
## F-statistic: 8.004 on 3 and 3573 DF, p-value: 2.579e-05
```

```
plot(stroke_num ~ avg_glucose_level, data = df_train_linear)
abline(lm.stroke.glucose)
```



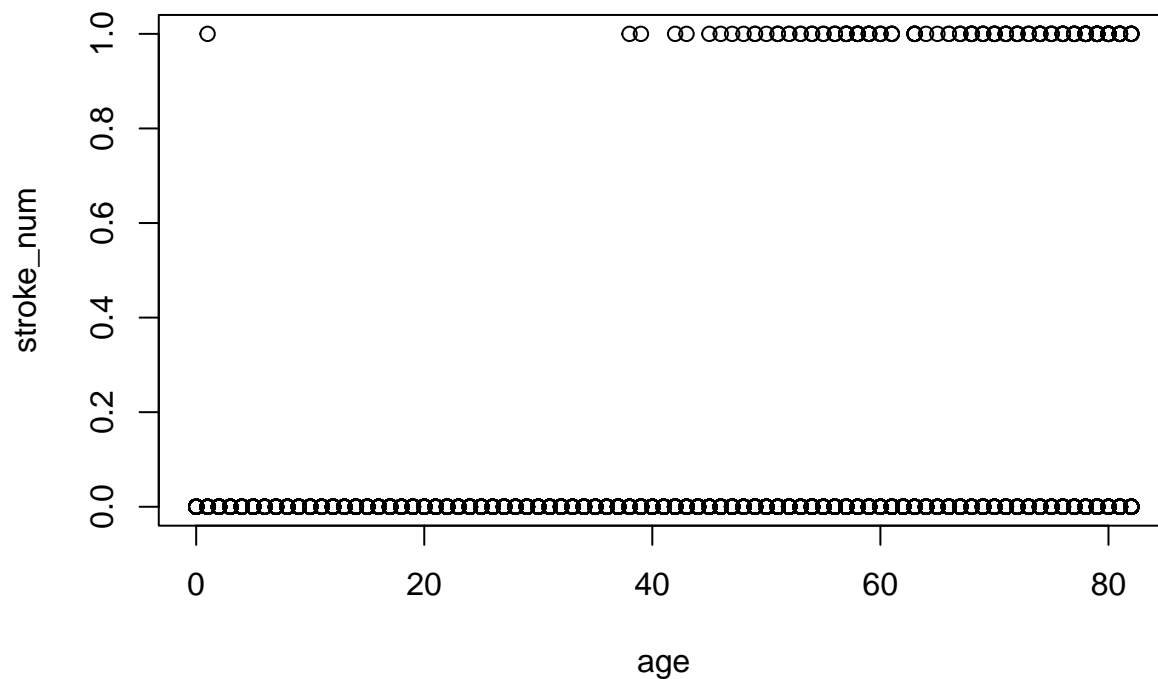
```
# fitting models for simple regression model
pred_var <-

lm.stroke.glucose <- lm(stroke_num ~ age + gender , data = df_train_linear)
summary(lm.stroke.smoking)
```

```
##
## Call:
```

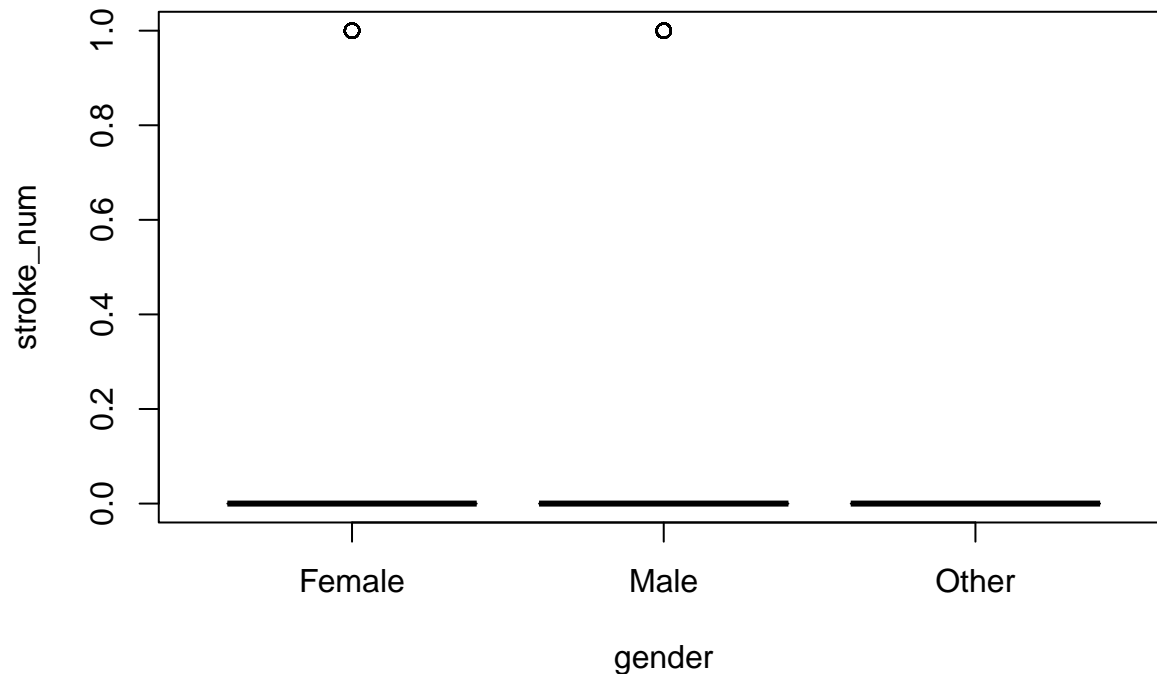
```
## lm(formula = stroke_num ~ smoking_status, data = df_train_linear)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.08548 -0.05244 -0.04989 -0.03145  0.96855
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)      0.085484   0.008801   9.713 < 2e-16 ***
## smoking_statusnever smoked -0.035597   0.010666  -3.338 0.000854 ***
## smoking_statussmokes    -0.033043   0.012818  -2.578 0.009981 **
## smoking_statusUnknown   -0.054032   0.011040  -4.894 1.03e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2191 on 3573 degrees of freedom
## Multiple R-squared:  0.006675,    Adjusted R-squared:  0.005841
## F-statistic: 8.004 on 3 and 3573 DF,  p-value: 2.579e-05
```

```
plot(stroke_num ~ age + gender, data = df_train_linear)
```



```
abline(lm.stroke.glucose)
```

```
## Warning in abline(lm.stroke.glucose): only using the first two of 4 regression
## coefficients
```

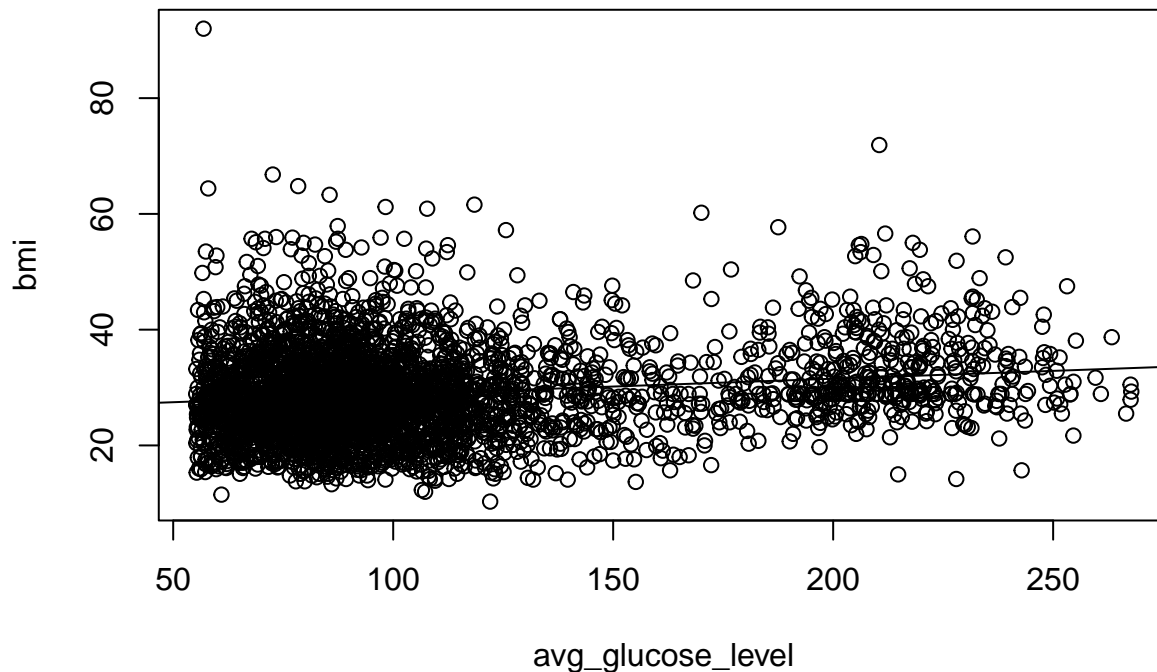


```
# fitting models for simple regression model
```

```
lm.stroke.bmi <- lm(bmi ~ avg_glucose_level, data = df_train_linear)
summary(lm.stroke.bmi)
```

```
##
## Call:
## lm(formula = bmi ~ avg_glucose_level, data = df_train_linear)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -19.116  -4.919  -0.754   3.807  64.349
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    26.109983   0.320458  81.477  <2e-16 ***
## avg_glucose_level  0.027086   0.002752   9.843  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.554 on 3575 degrees of freedom
## Multiple R-squared:  0.02639,    Adjusted R-squared:  0.02612
## F-statistic: 96.89 on 1 and 3575 DF,  p-value: < 2.2e-16
```

```
plot(bmi ~ avg_glucose_level, data = df_train_linear)
abline(lm.stroke.bmi)
```



6 Generalised Linear Model with family set to Poisson

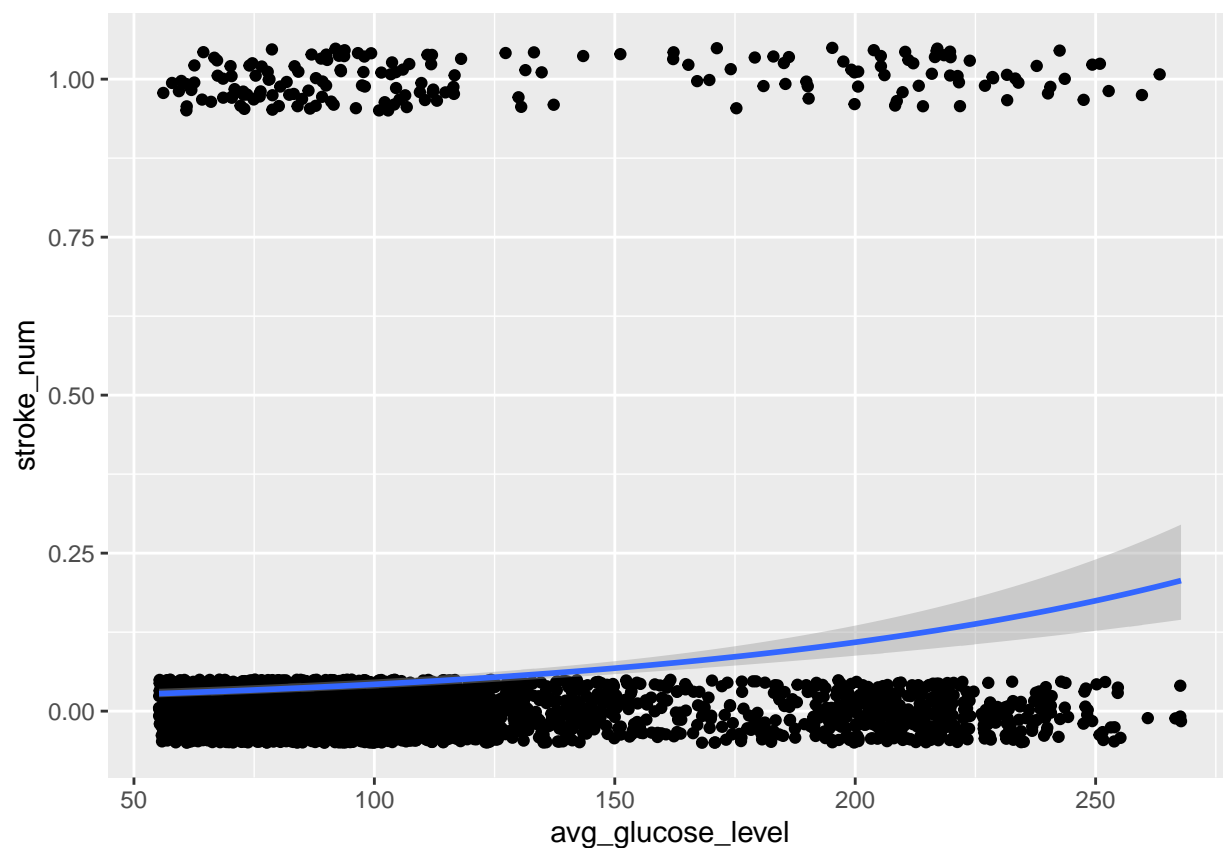
```
# test
glm.stroke.poisson <- glm(stroke_num ~ age + heart_disease + hypertension + avg_glucose_level,
family = "poisson",
data = df_train_linear)
summary(glm.stroke.poisson)
```

```
##
## Call:
## glm(formula = stroke_num ~ age + heart_disease + hypertension +
##      avg_glucose_level, family = "poisson", data = df_train_linear)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.0558  -0.3142  -0.1712  -0.0831   3.4861
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -7.384780   0.413043 -17.879  < 2e-16 ***
## age           0.066499   0.005875  11.319  < 2e-16 ***
## heart_disease  0.356522   0.186276   1.914  0.05563 .
## hypertension  0.296479   0.168541   1.759  0.07856 .
## avg_glucose_level 0.003448   0.001222   2.823  0.00476 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
```

```
## Null deviance: 1084.09 on 3576 degrees of freedom
## Residual deviance: 795.51 on 3572 degrees of freedom
## AIC: 1169.5
##
## Number of Fisher Scoring iterations: 7
```

```
#plot(stroke_num ~ age + heart_disease + hypertension + avg_glucose_level, data = df_train_linear)
#abline(glm.stroke_poisson)
ggplot(data = df_train_linear, aes(x = avg_glucose_level, y = stroke_num)) +
  geom_jitter(width = 0, height = 0.05) +
  geom_smooth(method = "glm", method.args = list(family = "poisson"))
```

```
## 'geom_smooth()' using formula 'y ~ x'
```



```
# Evaluating model fit poisson
```

```
# fitted(glm.stroke_poisson)
fitted.glm.stroke.poisson <- ifelse(fitted(glm.stroke.poisson) < 0.08, yes = 0, no = 1)
head(fitted.glm.stroke.poisson)
```

```
## 1 2 3 4 5 6
## 1 1 0 1 1 1
```

```
obs.fitted.comp.poisson <- data.frame(obs = df_train_linear$stroke_num, fitted = fitted.glm.stroke.poisson)
table(obs = obs.fitted.comp.poisson$obs, fit = obs.fitted.comp.poisson$fitted)
```

```
##      fit
## obs    0    1
##    0 2809  586
##    1   51  131
```

```
table(obs = obs.fitted.comp.poisson$obs, fit = obs.fitted.comp.poisson$fitted) %>%
  prop.table() %>%
  round(digits = 2)
```

```
##      fit
## obs    0    1
##    0 0.79 0.16
##    1 0.01 0.04
```

7 Generalised Linear Model with family set to Binomial

Since we are essentially dealing with a classification issue, using logistic regression in the form of a GLM with family set to “binomial” is the best method to apply out of all the models introduced so far. For this reason, we shall go into more detail here.

```
# Include all variables to start variable selection
# Plot all of them for visual analysis
glm.stroke.binomial <- glm(stroke ~ .,
  family = "binomial",
  data = df_train_linear)
```

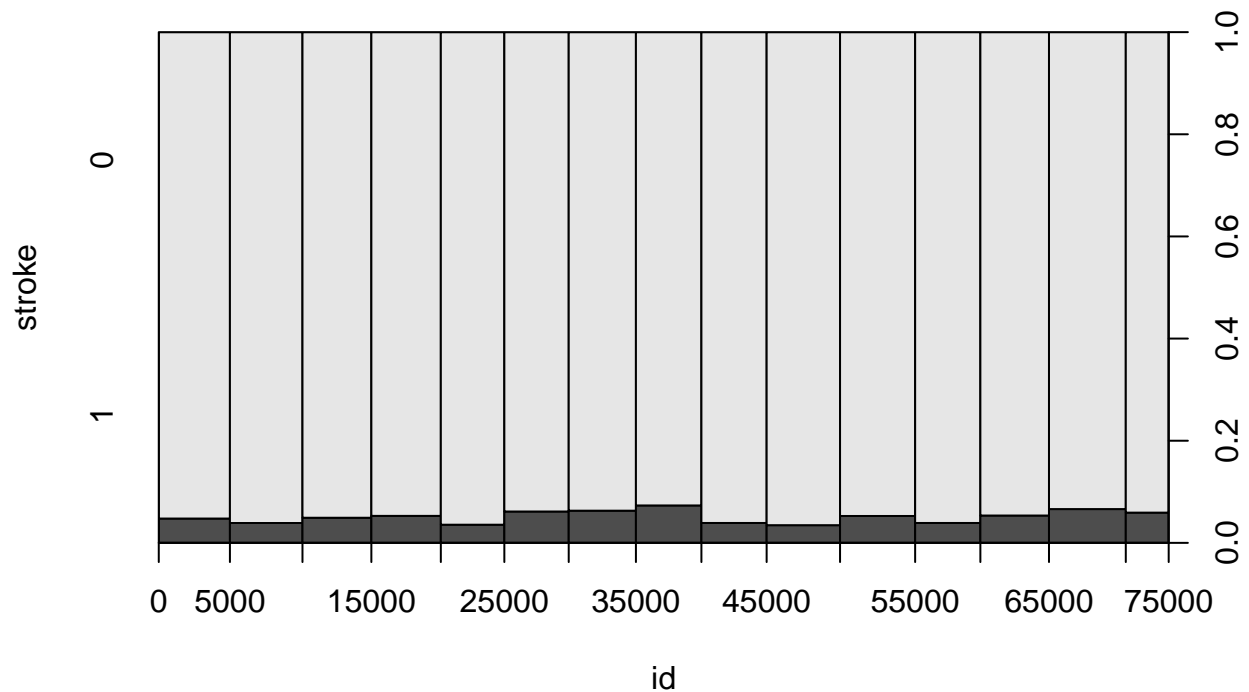
```
## Warning: glm.fit: algorithm did not converge
```

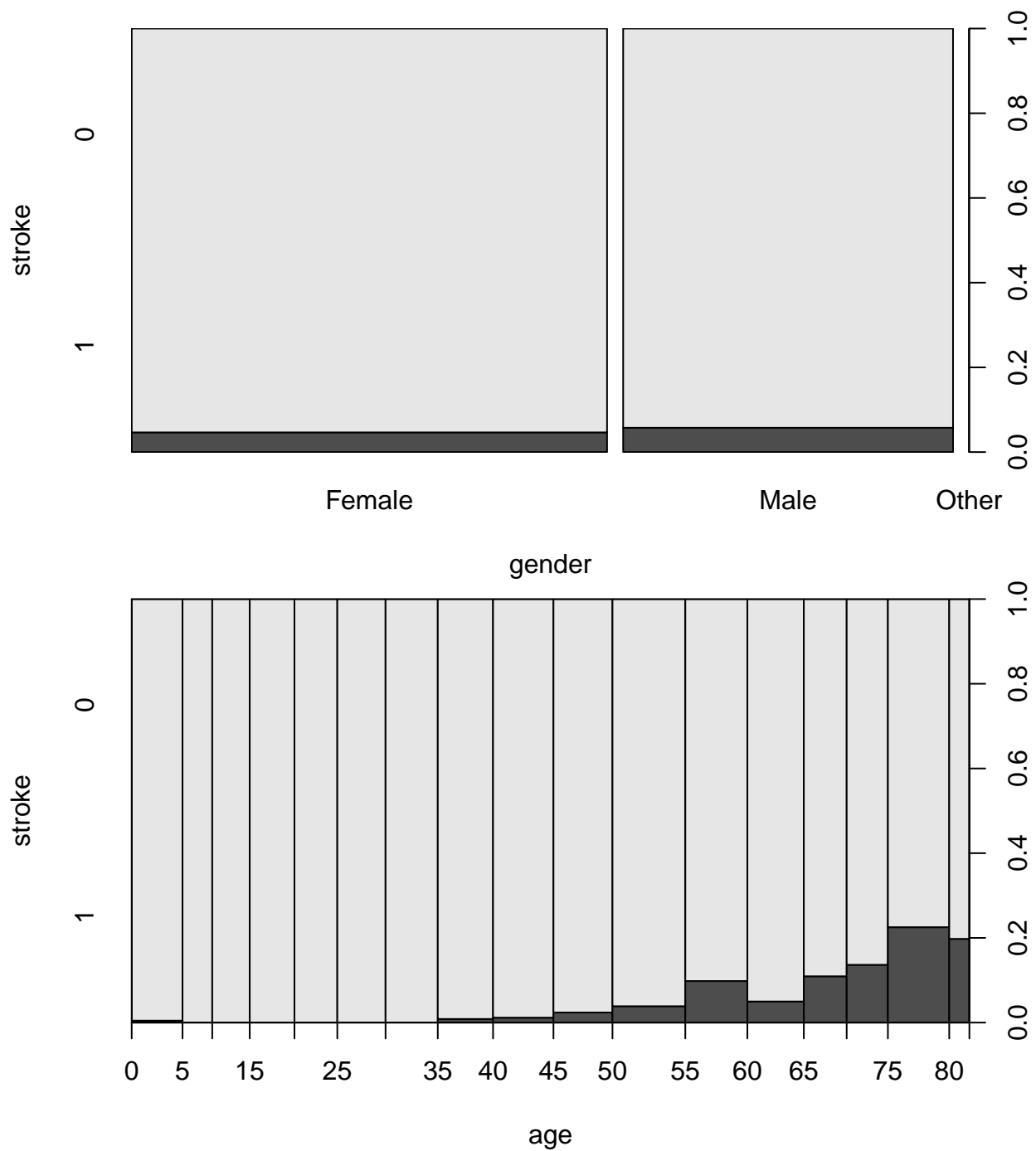
```
summary(glm.stroke.binomial)
```

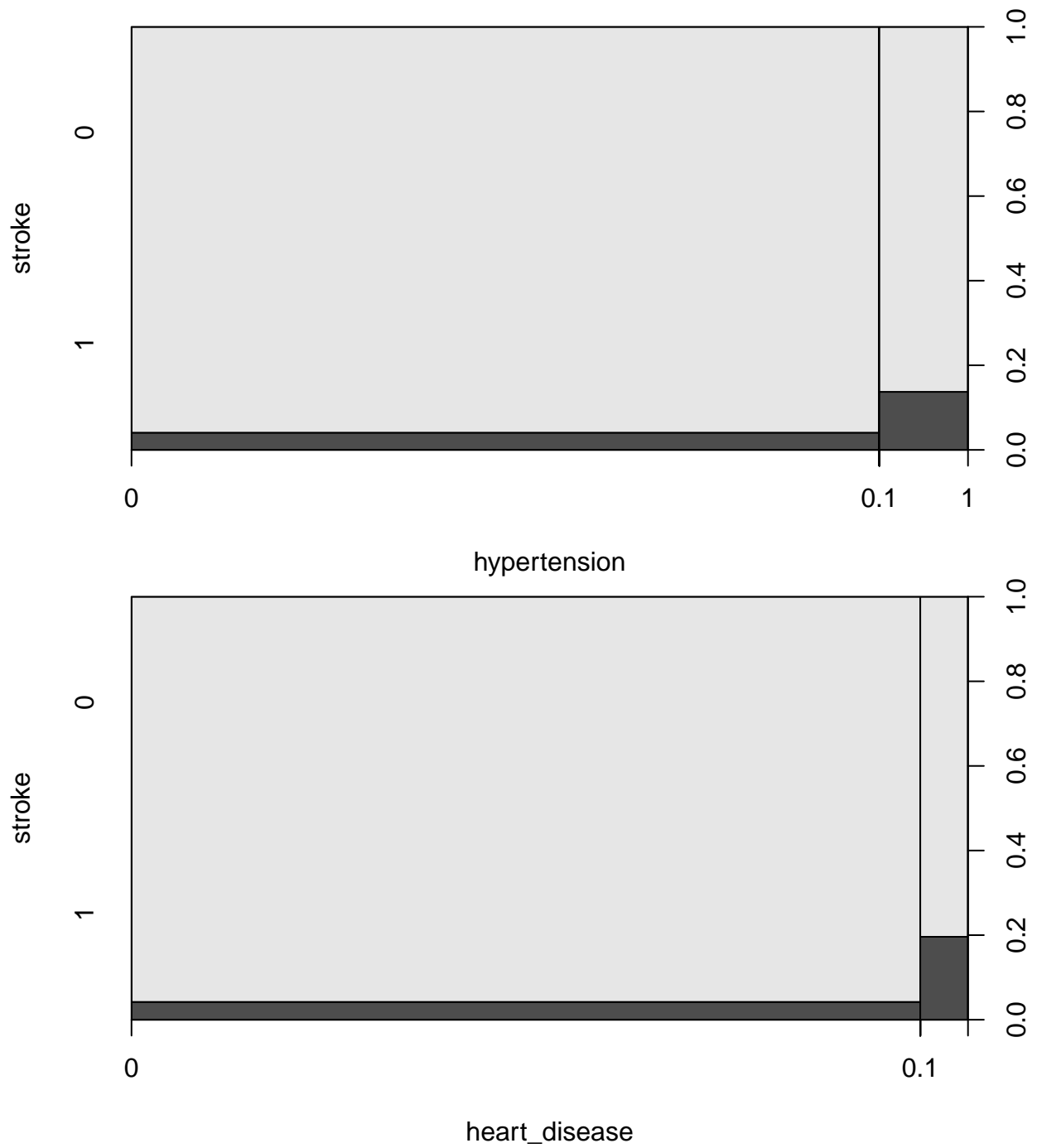
```
##
## Call:
## glm(formula = stroke ~ ., family = "binomial", data = df_train_linear)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -2.409e-06 -2.409e-06 -2.409e-06 -2.409e-06  2.409e-06
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.657e+01  3.576e+04  -0.001    0.999
## id          -2.854e-18  2.789e-01   0.000    1.000
## genderMale    1.062e-13  1.228e+04   0.000    1.000
## genderOther  -2.686e-13  3.569e+05   0.000    1.000
## age          -1.573e-14  4.548e+02   0.000    1.000
```

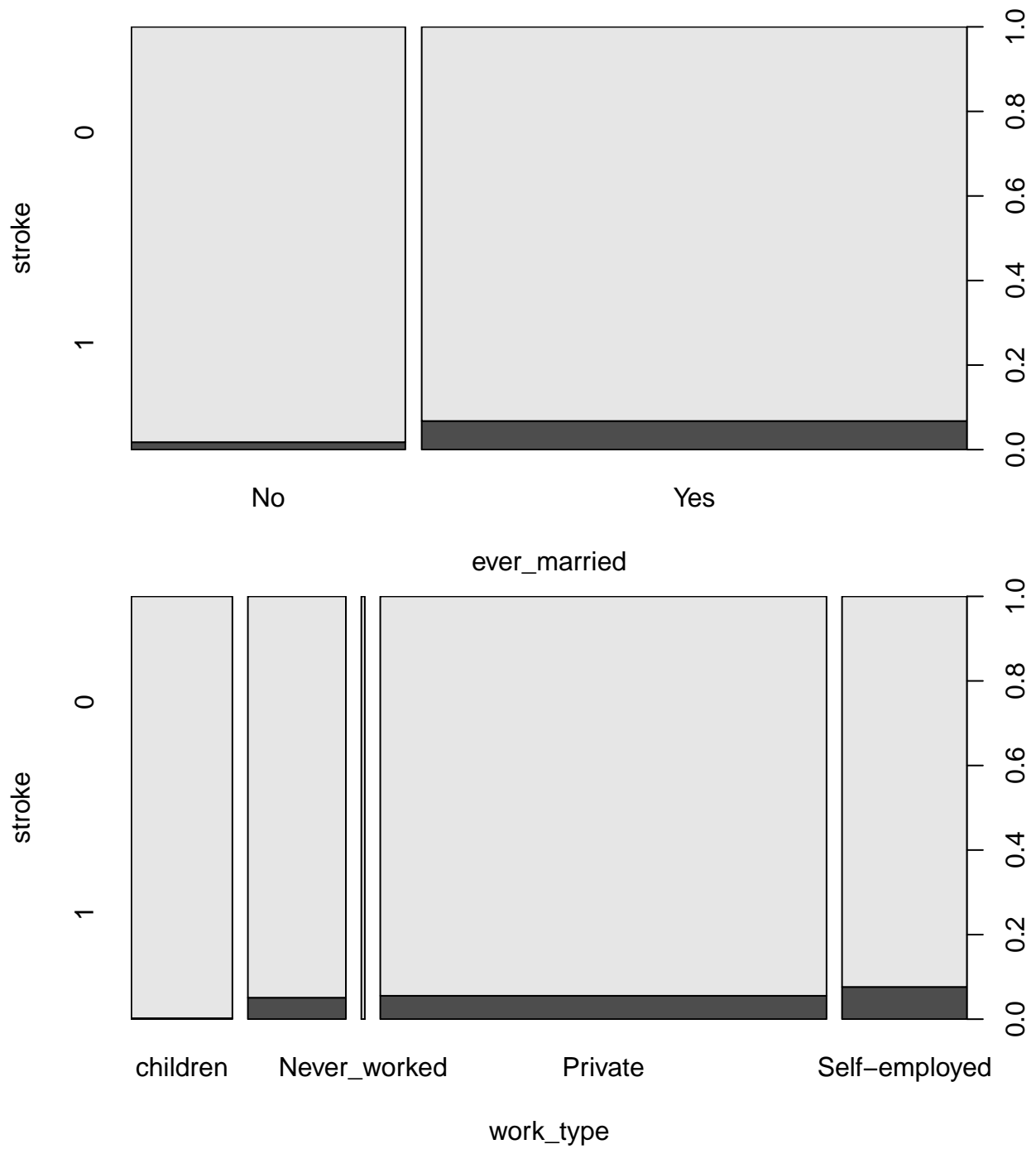
```
## hypertension          -4.872e-13  2.060e+04  0.000  1.000
## heart_disease         4.121e-12  2.725e+04  0.000  1.000
## ever_marriedYes       2.434e-13  1.780e+04  0.000  1.000
## work_typeGovt_job     3.477e-14  3.160e+04  0.000  1.000
## work_typeNever_worked -6.480e-14  9.134e+04  0.000  1.000
## work_typePrivate      2.136e-13  2.654e+04  0.000  1.000
## work_typeSelf-employed -5.576e-14  3.226e+04  0.000  1.000
## residence_typeUrban   2.297e-13  1.193e+04  0.000  1.000
## avg_glucose_level     5.150e-15  1.367e+02  0.000  1.000
## bmi                   1.133e-14  8.838e+02  0.000  1.000
## smoking_statusnever smoked -3.193e-13  1.768e+04  0.000  1.000
## smoking_statussmokes  -1.892e-13  2.109e+04  0.000  1.000
## smoking_statusUnknown -2.540e-13  1.978e+04  0.000  1.000
## stroke_num            5.313e+01  2.850e+04  0.002  0.999
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 1.4387e+03  on 3576  degrees of freedom
## Residual deviance: 2.0752e-08  on 3558  degrees of freedom
## AIC: 38
##
## Number of Fisher Scoring iterations: 25
```

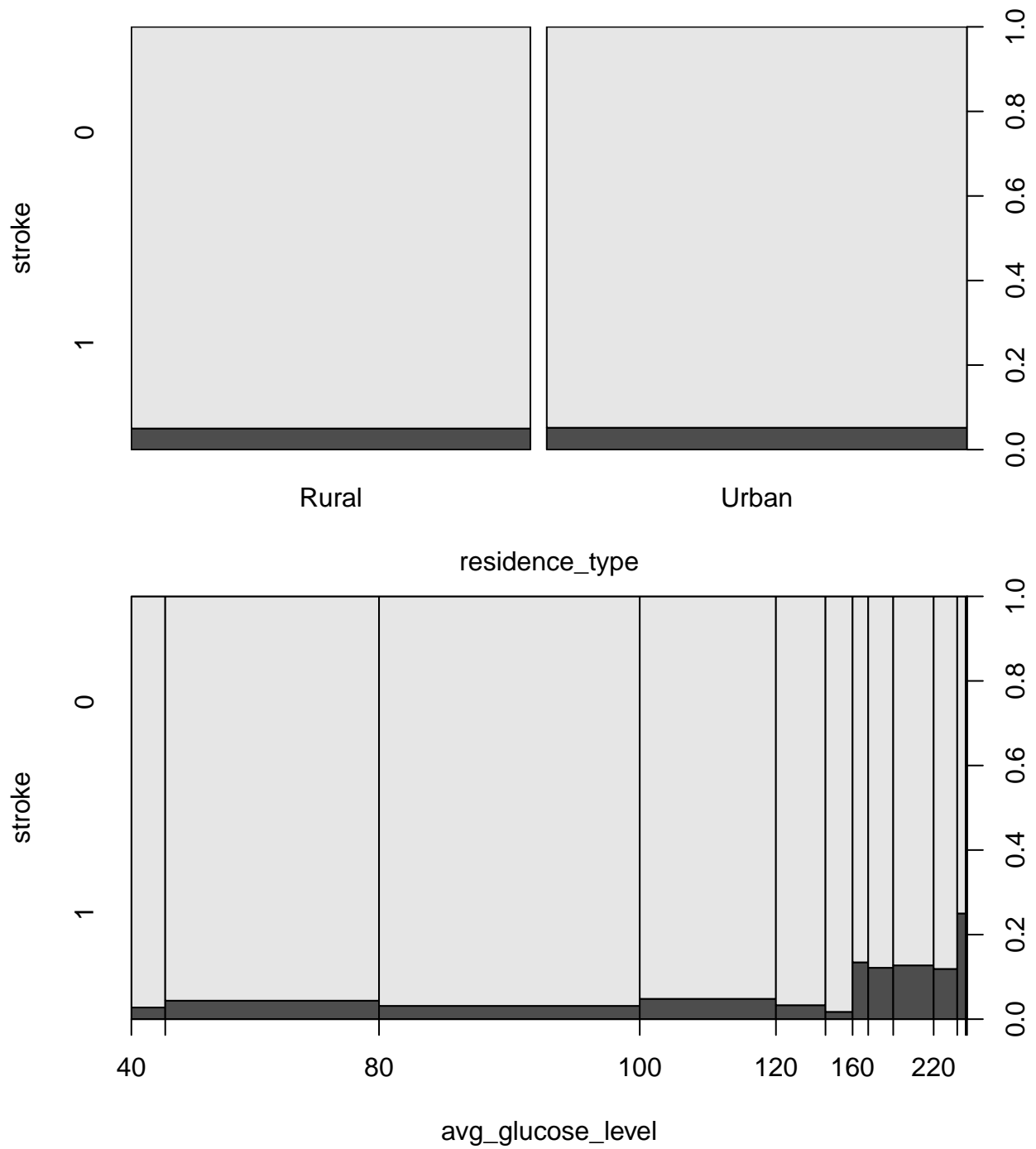
```
plot(stroke ~ ., data = df_train_linear)
```

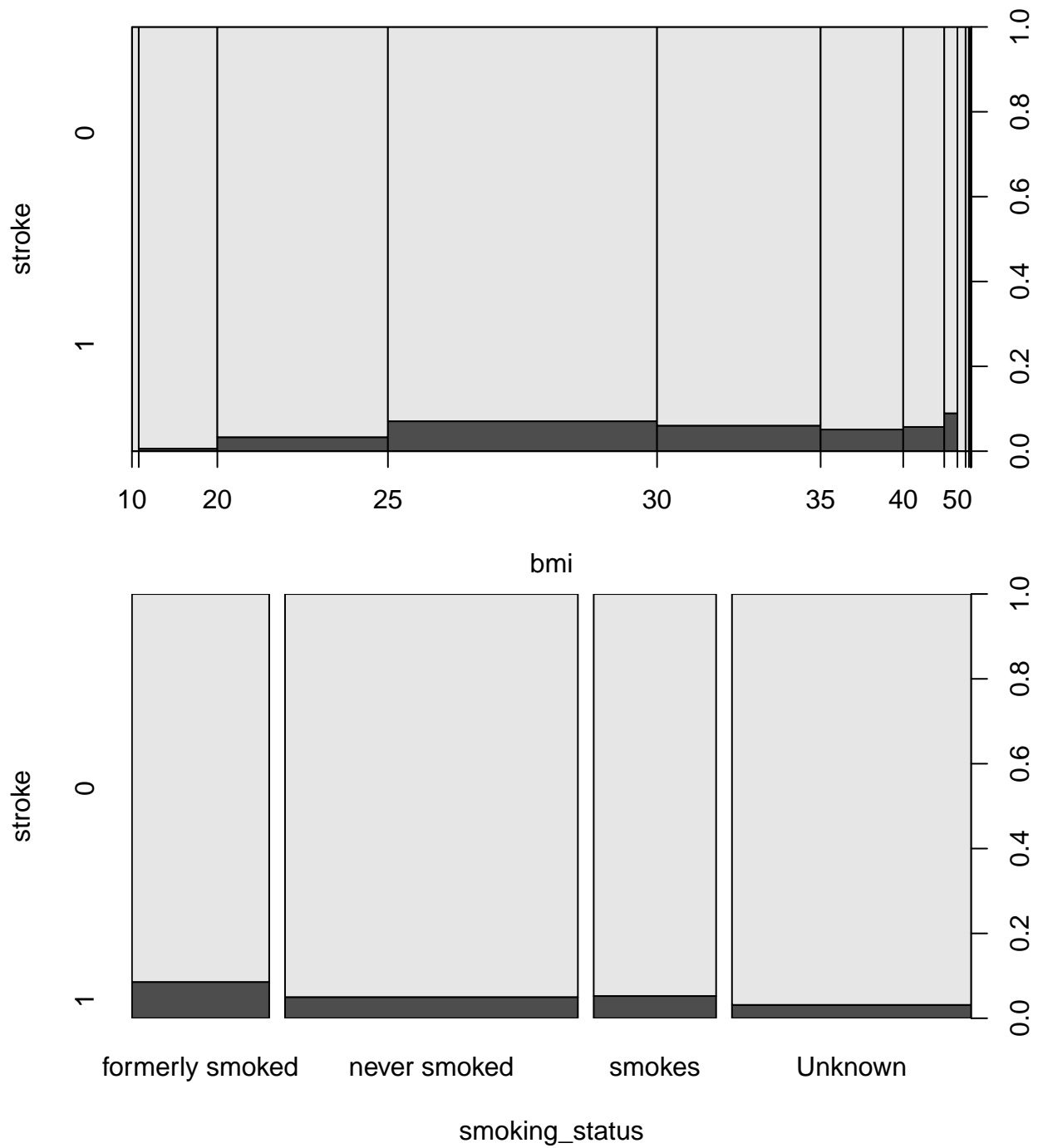






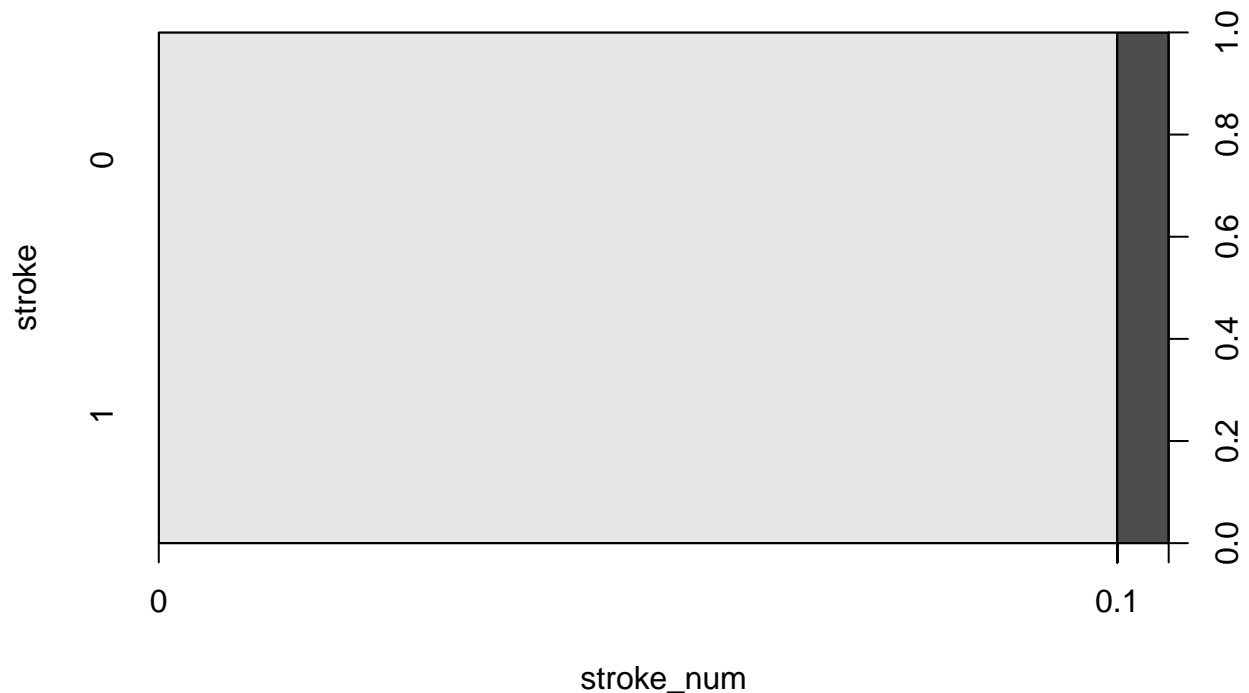






```
abline(glm.stroke.binomial)
```

```
## Warning in abline(glm.stroke.binomial): only using the first two of 19
## regression coefficients
```

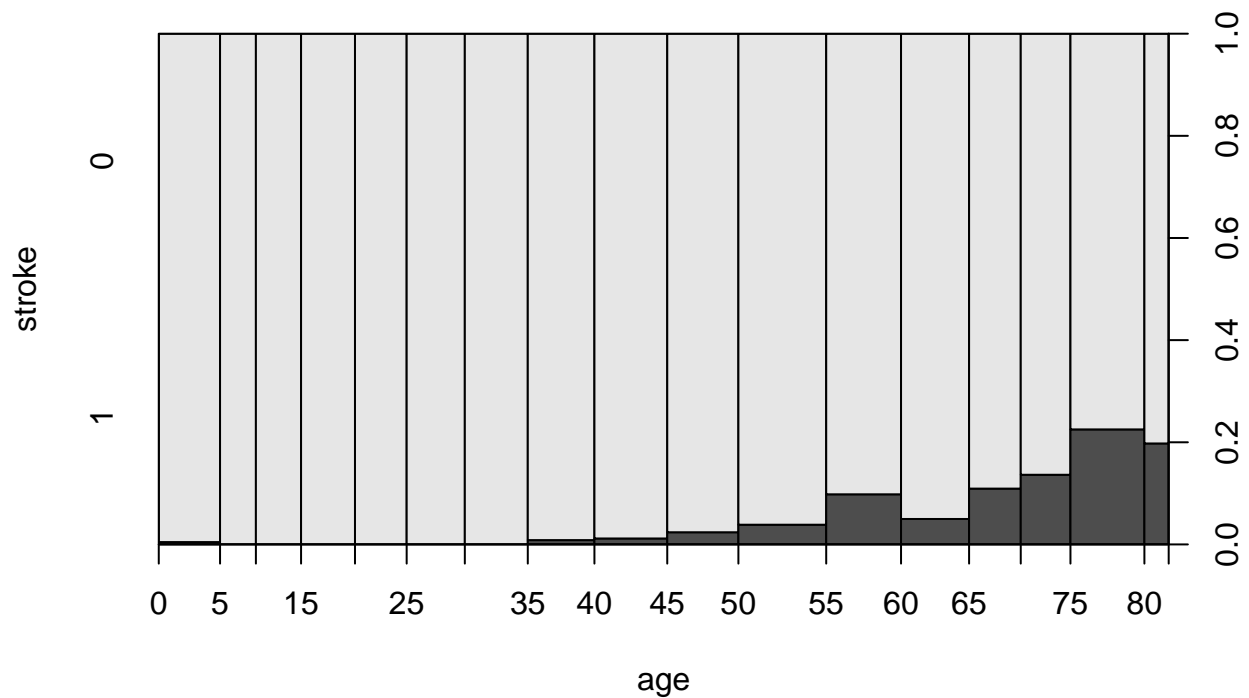


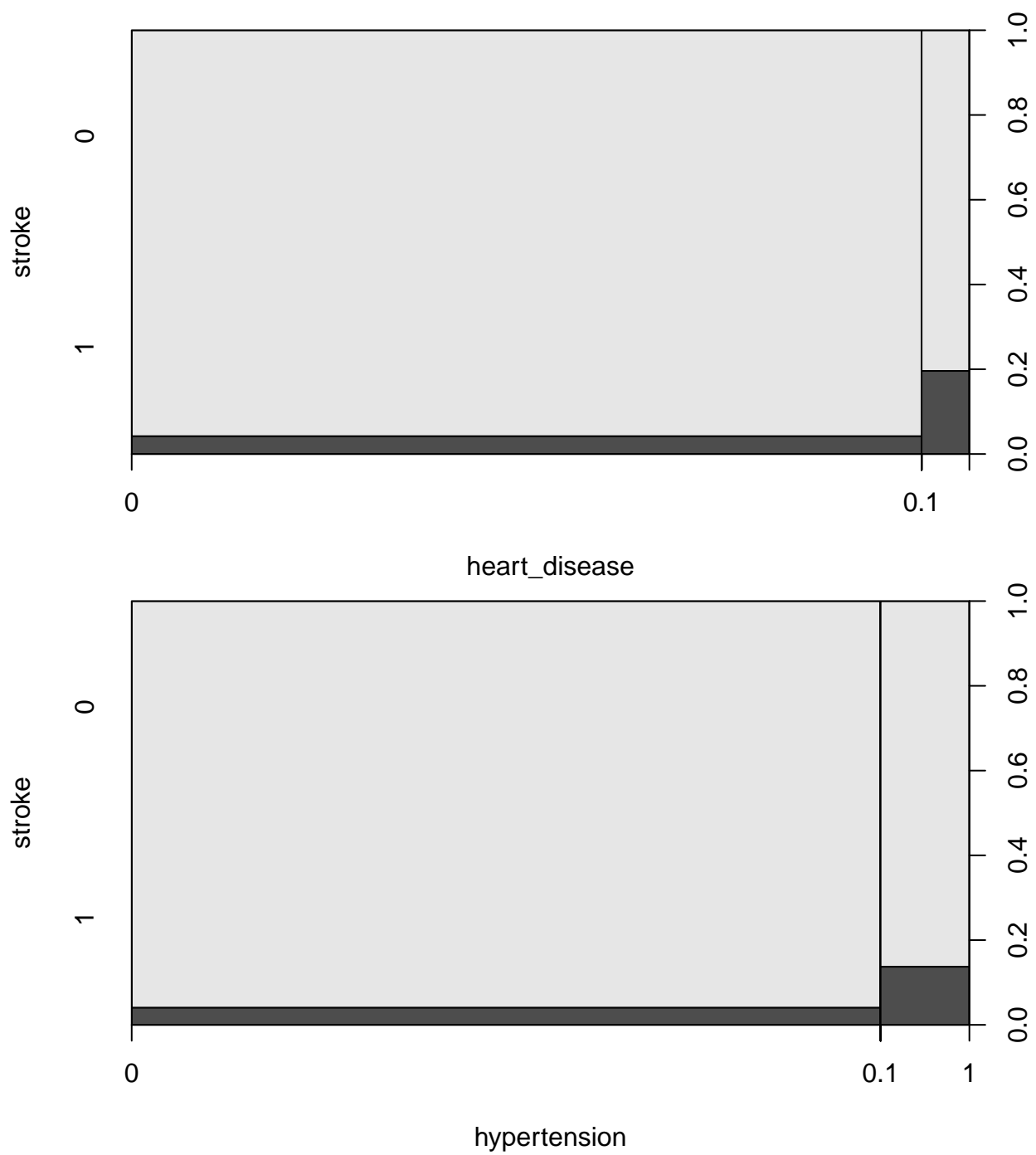
```
# First iteration with parameters chosen from intuitive domain knowledge and exploratory analysis of data
glm.stroke.binomial <- glm(stroke ~ age + heart_disease + hypertension + work_type + avg_glucose_level +
family = "binomial",
data = df_train_linear)
summary(glm.stroke.binomial)
```

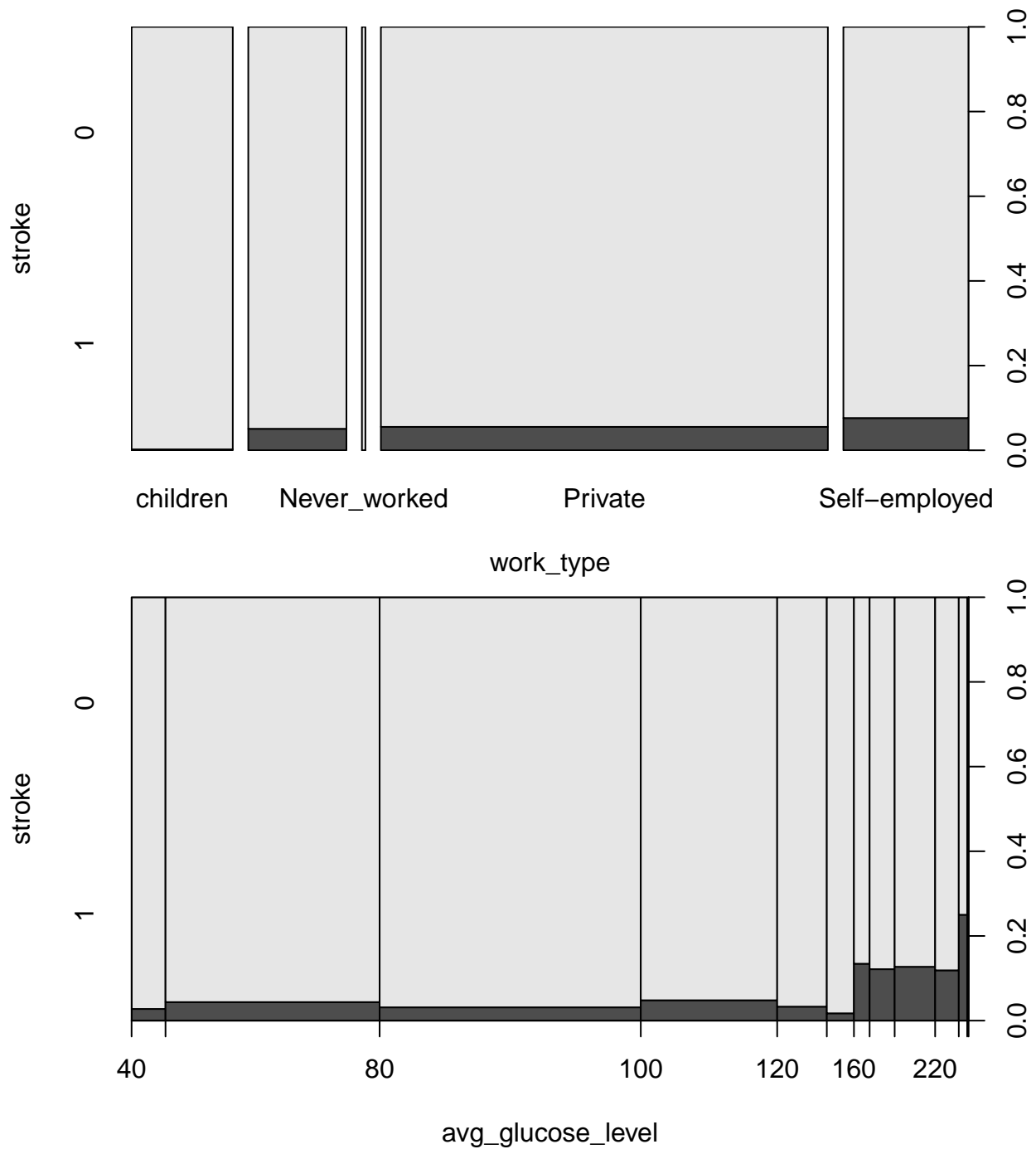
```
##
## Call:
## glm(formula = stroke ~ age + heart_disease + hypertension + work_type +
##     avg_glucose_level + smoking_status, family = "binomial",
##     data = df_train_linear)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.1696  -0.3241  -0.1595  -0.0771   3.6785
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -7.037953   1.036288  -6.792 1.11e-11 ***
## age              0.078346   0.006958  11.260 < 2e-16 ***
## heart_disease    0.431469   0.213875   2.017  0.0437 *
## hypertension     0.378996   0.189527   2.000  0.0455 *
## work_typeGovt_job -1.003706   1.105068  -0.908  0.3637
## work_typeNever_worked -10.077332 362.934034  -0.028  0.9778
## work_typePrivate  -0.818567   1.088363  -0.752  0.4520
## work_typeSelf-employed -1.382649   1.112789  -1.243  0.2140
## avg_glucose_level  0.003936   0.001360   2.894  0.0038 **
## smoking_statusnever smoked -0.186796   0.203452  -0.918  0.3586
## smoking_statussmokes  0.032194   0.256225   0.126  0.9000
## smoking_statusUnknown -0.081986   0.243356  -0.337  0.7362
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##    Null deviance: 1438.7  on 3576  degrees of freedom
## Residual deviance: 1117.5  on 3565  degrees of freedom
## AIC: 1141.5
##
## Number of Fisher Scoring iterations: 14
```

```
plot(stroke ~ age + heart_disease + hypertension + work_type + avg_glucose_level + smoking_status, data = data)
```

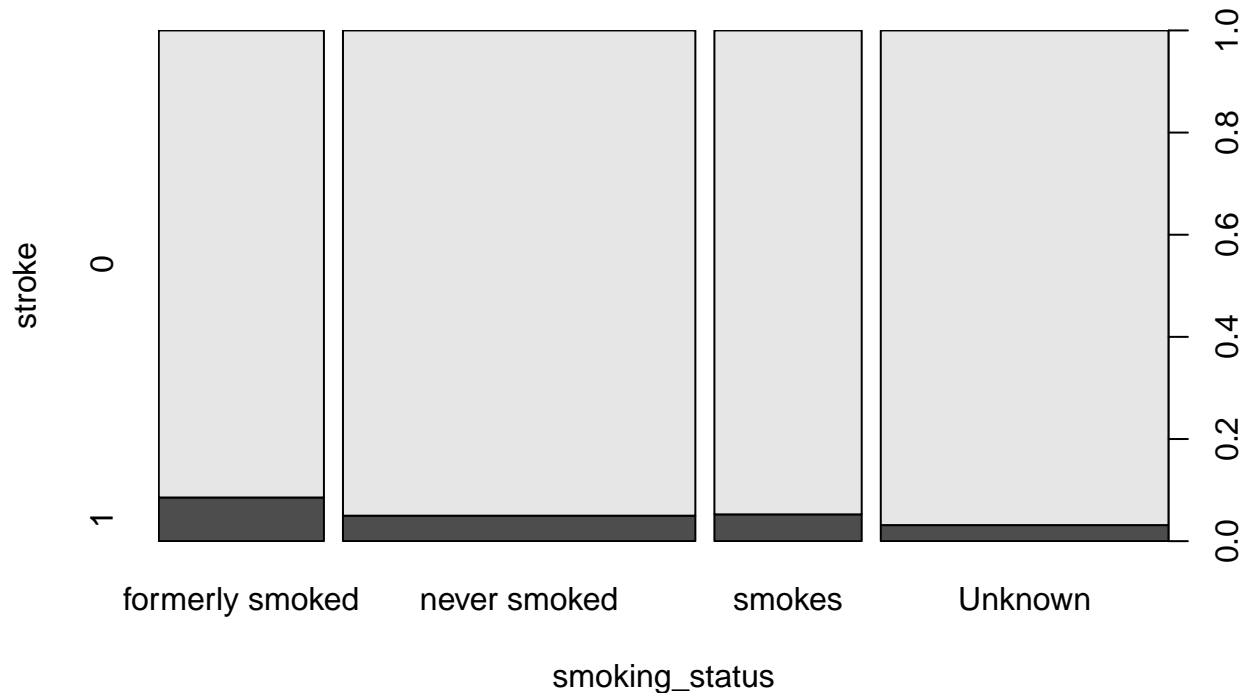






```
abline(glm.stroke.binomial)
```

```
## Warning in abline(glm.stroke.binomial): only using the first two of 12
## regression coefficients
```

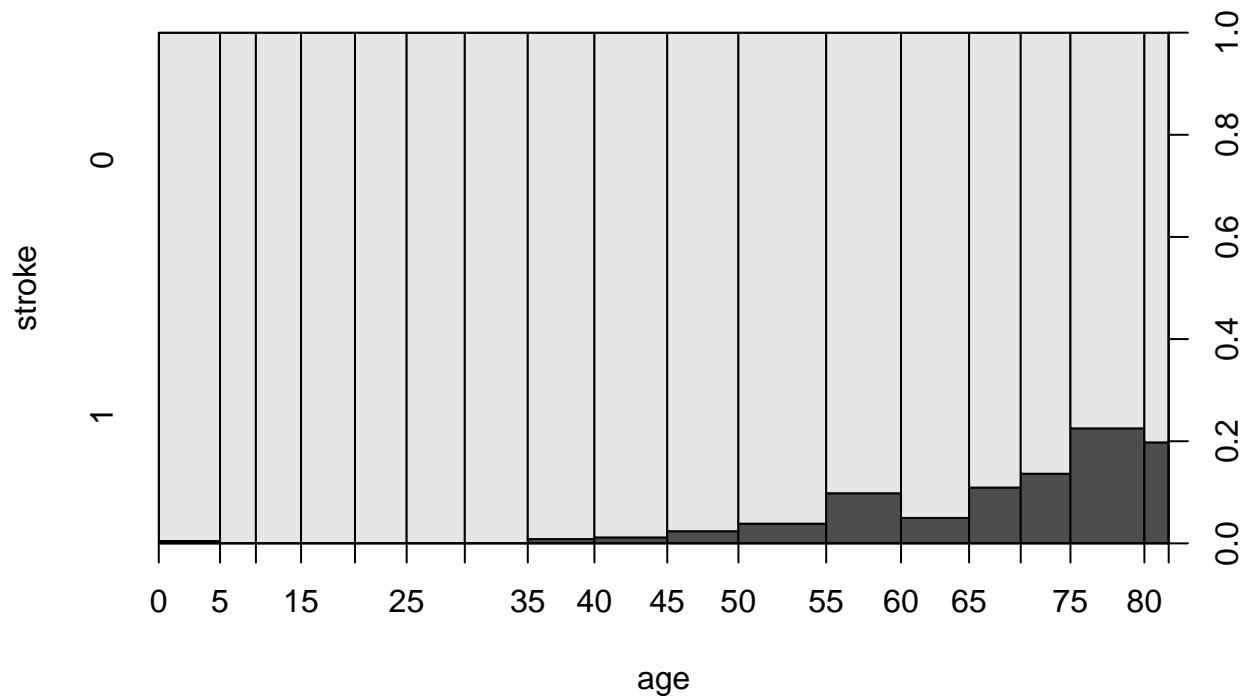


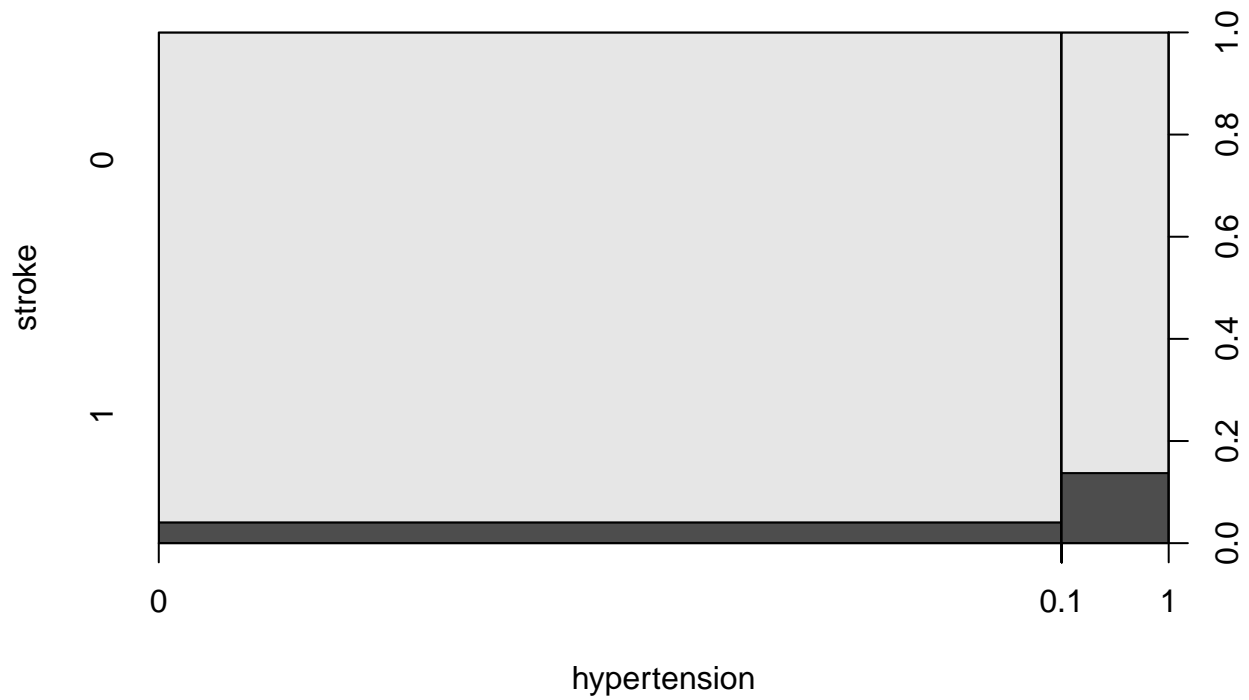
```
# second iteration only keeping statistically relevant parameters from previous model
glm.stroke.binomial.2 <- glm(stroke_num ~ age + heart_disease + hypertension + avg_glucose_level,
family = "binomial",
data = df_train_linear)
summary(glm.stroke.binomial.2)
```

```
##
## Call:
## glm(formula = stroke_num ~ age + heart_disease + hypertension +
##     avg_glucose_level, family = "binomial", data = df_train_linear)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.1183  -0.3237  -0.1671  -0.0764   3.8413
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -7.739236   0.443953 -17.433  <2e-16 ***
## age           0.072374   0.006312  11.467  <2e-16 ***
## heart_disease  0.465121   0.211312   2.201   0.0277 *
## hypertension   0.362843   0.187103   1.939   0.0525 .
## avg_glucose_level 0.004117   0.001351   3.048   0.0023 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 1438.7  on 3576  degrees of freedom
## Residual deviance: 1127.6  on 3572  degrees of freedom
## AIC: 1137.6
##
```

```
## Number of Fisher Scoring iterations: 7
```

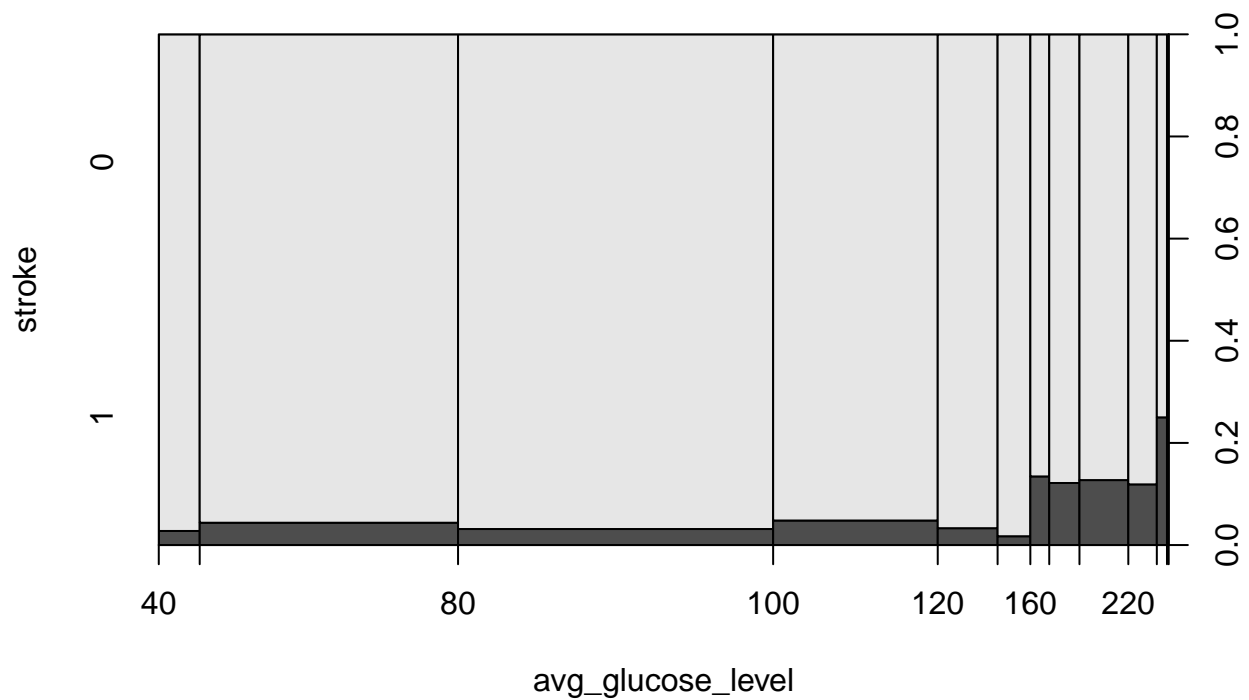
```
plot(stroke ~ age + heart_disease + hypertension + avg_glucose_level, data = df_train_linear)
```





```
abline(glm.stroke.binomial.2)
```

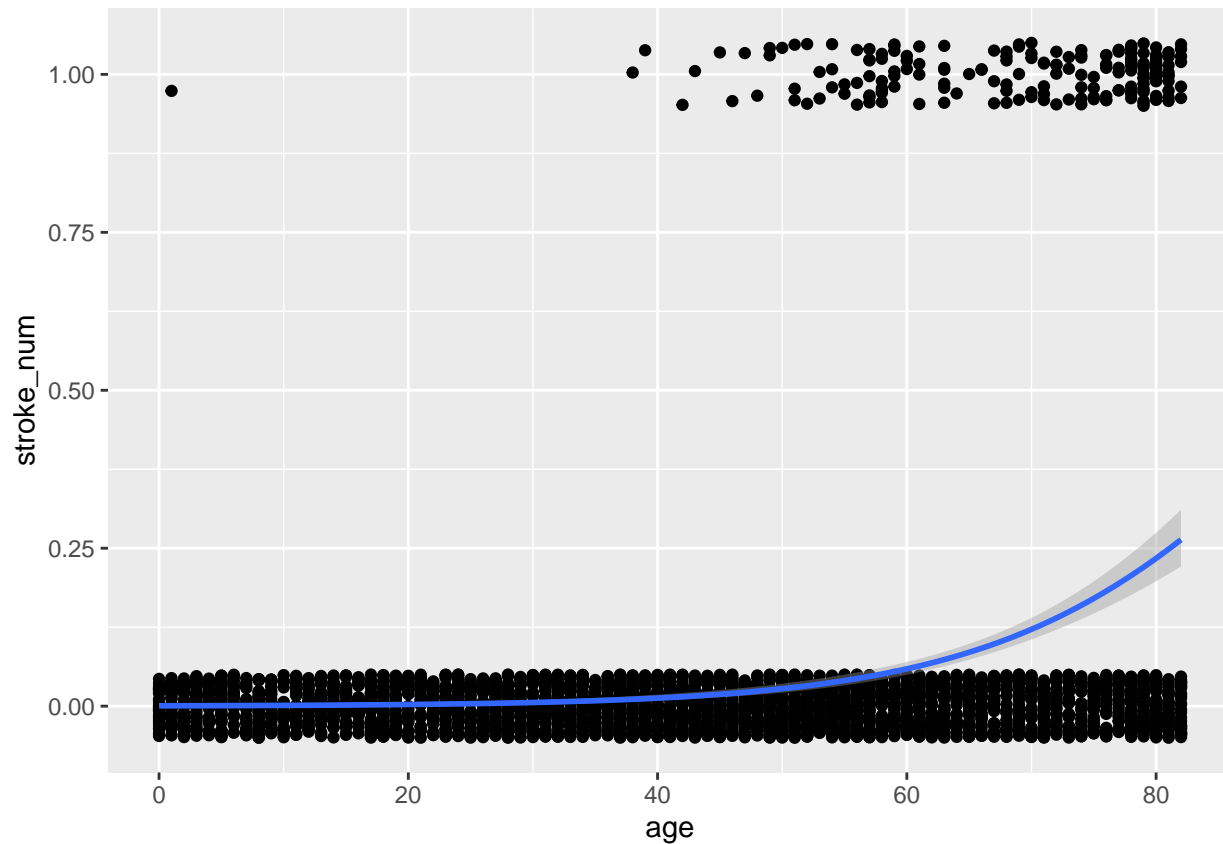
```
## Warning in abline(glm.stroke.binomial.2): only using the first two of 5
## regression coefficients
```



```
pred <- predict(glm.stroke.binomial.2)
# pred
```

```
ggplot(data = df_train_linear, aes(x = age, y = stroke_num)) +
  geom_jitter(width = 0, height = 0.05) +
  geom_smooth(method = "glm", method.args = list(family = "binomial"))
```

```
## 'geom_smooth()' using formula 'y ~ x'
```



```
# Evaluating model fit using fitted()
```

```
# fitted(glm.stroke.binomial.2)
```

```
fitted.glm.stroke.binomial.2 <- ifelse(fitted(glm.stroke.binomial.2) < 0.2, yes = 0, no = 1)
```

```
head(fitted.glm.stroke.binomial.2)
```

```
## 1 2 3 4 5 6
```

```
## 0 1 0 1 1 1
```

```
obs.fitted.comp <- data.frame(obs = df_train_linear$stroke_num, fitted = fitted.glm.stroke.binomial.2)
```

```
table(obs = obs.fitted.comp$obs, fit = obs.fitted.comp$fitted)
```

```
##      fit
```

```
## obs    0    1
```

```
##    0 3235 160
```

```
##    1  130  52
```

```
table(obs = obs.fitted.comp$obs, fit = obs.fitted.comp$fitted) %>%
  prop.table() %>%
  round(digits = 2)
```

```
##      fit
## obs    0    1
##    0 0.90 0.04
##    1 0.04 0.01
```

```
# Evaluating model fit using predict
```

```
# predict(glm.stroke.binomial.2, df_test_linear, type = "response")
```

```
# predicted(glm.stroke.binomial.2)
```

```
predicted.glm.stroke.binomial.2 <- ifelse(predict(glm.stroke.binomial.2, df_test_linear, type = "response") > 0.5, 1, 0)
head(predicted.glm.stroke.binomial.2)
```

```
## 1 2 3 4 5 6
## 0 0 0 0 0 0
```

```
obs.predicted.comp <- data.frame(obs = df_test_linear$stroke_num, predicted = predicted.glm.stroke.binomial.2)
```

```
table(obs = obs.predicted.comp$obs, fit = obs.predicted.comp$predicted)
```

```
##      fit
## obs    0    1
##    0 1412  54
##    1   54  13
```

```
table(obs = obs.predicted.comp$obs, fit = obs.predicted.comp$predicted) %>%
  prop.table() %>%
  round(digits = 2)
```

```
##      fit
## obs    0    1
##    0 0.92 0.04
##    1 0.04 0.01
```

8 Generalised Additive Model

```
library(mgcv)

gam.stroke <- gam(stroke ~ smoking_status + s(bmi),
  family = "binomial",
  data = df_train_linear)
summary(gam.stroke)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## stroke ~ smoking_status + s(bmi)
##
## Parametric coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -2.6671     0.1636 -16.302 < 2e-16 ***
## smoking_statusnever smoked  -0.5438     0.1924  -2.827  0.00470 **
## smoking_statussmokes      -0.5377     0.2398  -2.243  0.02492 *
## smoking_statusUnknown     -0.7270     0.2298  -3.164  0.00155 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##               edf Ref.df Chi.sq  p-value
## s(bmi) 4.745  5.588  26.22 0.000161 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.0132   Deviance explained =  4.5%
## UBRE = -0.61102   Scale est. = 1           n = 3577
```

```
plot(stroke ~ smoking_status, data = df_train_linear)
abline(gam.stroke)
```

```
## Warning in abline(gam.stroke): only using the first two of 13 regression
## coefficients
```



9 Neural Network Yves

10 Support Vector Machine (Larissa)

Stroke Data Classification using a Support Vector Machine.

```
#create train and test data set for SVM Model with a split 70 (training) / 30 (test)
set.seed(7406)
n=dim(stroke_data[1]) # number of observations in dataset
n_train=0.70*n # training set is 70%
flag = sort(sample(1:n, size=n_train, replace=FALSE))

## Warning in 1:n: numerical expression has 2 elements: only the first used

#use all parameters without first column id
df_train_svm = stroke_data[flag,]
df_test_svm = stroke_data[-flag,]

ytrain = df_train_svm$stroke
ytest = df_test_svm$stroke

table(df_train_svm$stroke)

##
##      0      1
## 3395  182

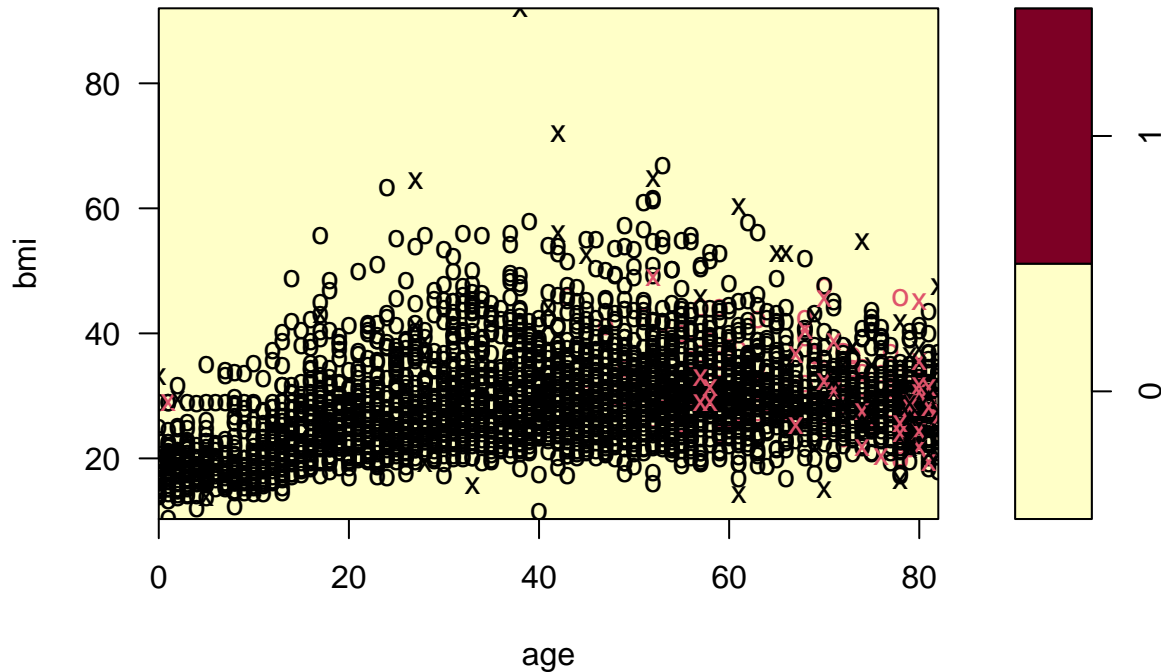
svm_model <- svm(stroke ~ ., data = df_train_svm, type = "C-classification", kernel = "radial")
summary(svm_model)

##
## Call:
## svm(formula = stroke ~ ., data = df_train_svm, type = "C-classification",
##      kernel = "radial")
##
##
## Parameters:
##   SVM-Type:  C-classification
## SVM-Kernel:  radial
##      cost:   1
##
## Number of Support Vectors:  95
##
## ( 31 64 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```



```
plot(svm_model, data = df_train_svm, bmi ~ age, slice = list(avg_glucose_level = 3))
```

SVM classification plot



```
#confusion matrix for training error
svm_training_prediction <- predict(svm_model, newdata = df_train_svm)
svm_training_error <- mean(svm_training_prediction != ytrain)
confusionMatrix(svm_training_prediction, df_train_svm$stroke)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##           0 3395    0
##           1    0 182
##
##           Accuracy : 1
##           95% CI : (0.999, 1)
##           No Information Rate : 0.9491
##           P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 1
##
##           McNemar's Test P-Value : NA
##
##           Sensitivity : 1.0000
##           Specificity : 1.0000
##           Pos Pred Value : 1.0000
##           Neg Pred Value : 1.0000
##           Prevalence : 0.9491
```

```
##          Detection Rate : 0.9491
##    Detection Prevalence : 0.9491
##      Balanced Accuracy : 1.0000
##
##      'Positive' Class : 0
##
```

```
svm_training_error
```

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

```
confusionMatrix(svm_training_prediction,df_train_svm$stroke)
```

```
## Confusion Matrix and Statistics
##
##          Reference
## Prediction    0    1
##          0 3395    0
##          1    0 182
##
##          Accuracy : 1
##          95% CI : (0.999, 1)
##    No Information Rate : 0.9491
##    P-Value [Acc > NIR] : < 2.2e-16
##
##          Kappa : 1
##
##    Mcnemar's Test P-Value : NA
##
##          Sensitivity : 1.0000
##          Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##          Prevalence : 0.9491
##          Detection Rate : 0.9491
##    Detection Prevalence : 0.9491
##          Balanced Accuracy : 1.0000
##
##      'Positive' Class : 0
##
```

```
#confusion matrix for test data
```

```
svm_prediction <- predict(svm_model, newdata = df_test_svm)
svm_test_error <- mean(svm_prediction != ytest)
confusionMatrix(svm_prediction,df_test_svm$stroke)
```

```
## Confusion Matrix and Statistics
##
##          Reference
## Prediction    0    1
##          0 1466    0
##          1    0   67
```

```
##
##           Accuracy : 1
##           95% CI : (0.9976, 1)
##    No Information Rate : 0.9563
##    P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##    Sensitivity : 1.0000
##    Specificity : 1.0000
##    Pos Pred Value : 1.0000
##    Neg Pred Value : 1.0000
##    Prevalence : 0.9563
##    Detection Rate : 0.9563
##    Detection Prevalence : 0.9563
##    Balanced Accuracy : 1.0000
##
##    'Positive' Class : 0
##
```

```
svm_test_error
```

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

```
#tune parameters for svm
```

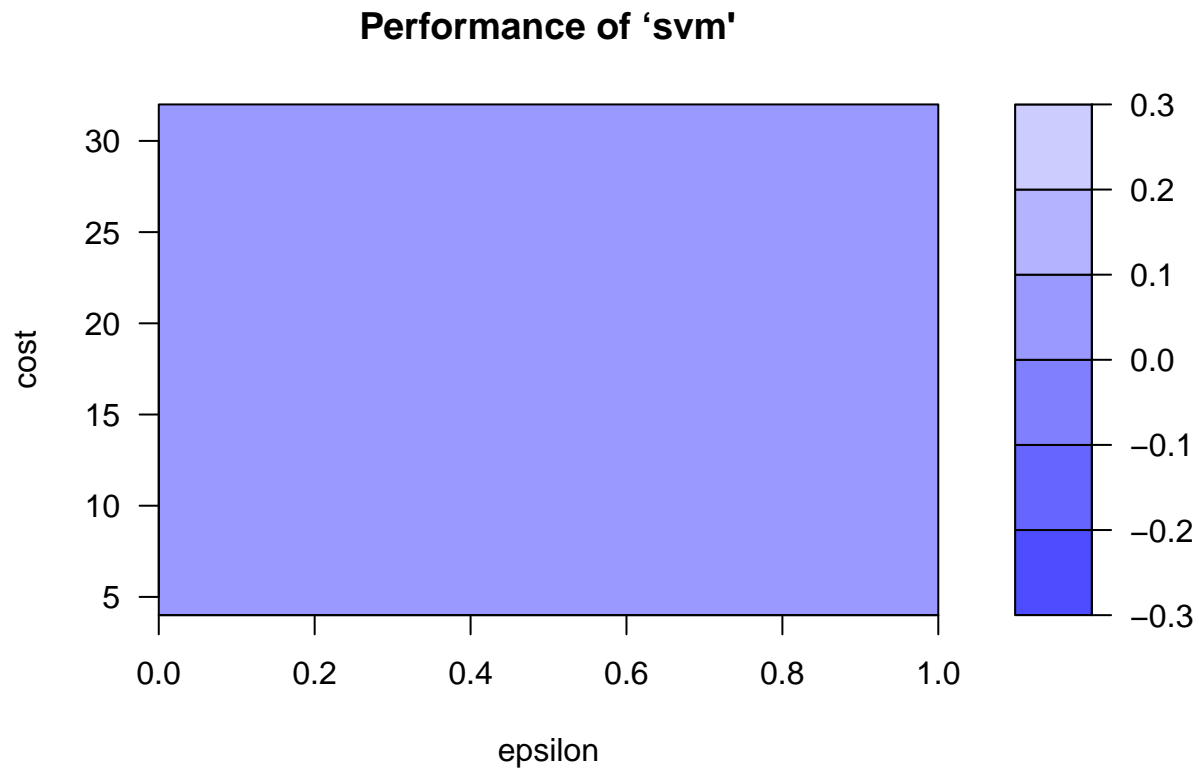
```
set.seed(123)
```

```
tuned_svm <- tune(svm, stroke ~ . , data = df_train_svm, ranges = list(epsilon = seq(0, 1, 0.1), cost =
```

```
tuned_svm
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   epsilon cost
##     0      4
##
## - best performance: 0
```

```
plot(tuned_svm)
```



```
summary(tuned_svm)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##   epsilon cost
##     0     4
##
## - best performance: 0
##
## - Detailed performance results:
##   epsilon cost error dispersion
## 1    0.0    4    0            0
## 2    0.1    4    0            0
## 3    0.2    4    0            0
## 4    0.3    4    0            0
## 5    0.4    4    0            0
## 6    0.5    4    0            0
## 7    0.6    4    0            0
## 8    0.7    4    0            0
## 9    0.8    4    0            0
## 10   0.9    4    0            0
## 11   1.0    4    0            0
## 12   0.0    8    0            0
## 13   0.1    8    0            0
## 14   0.2    8    0            0
```

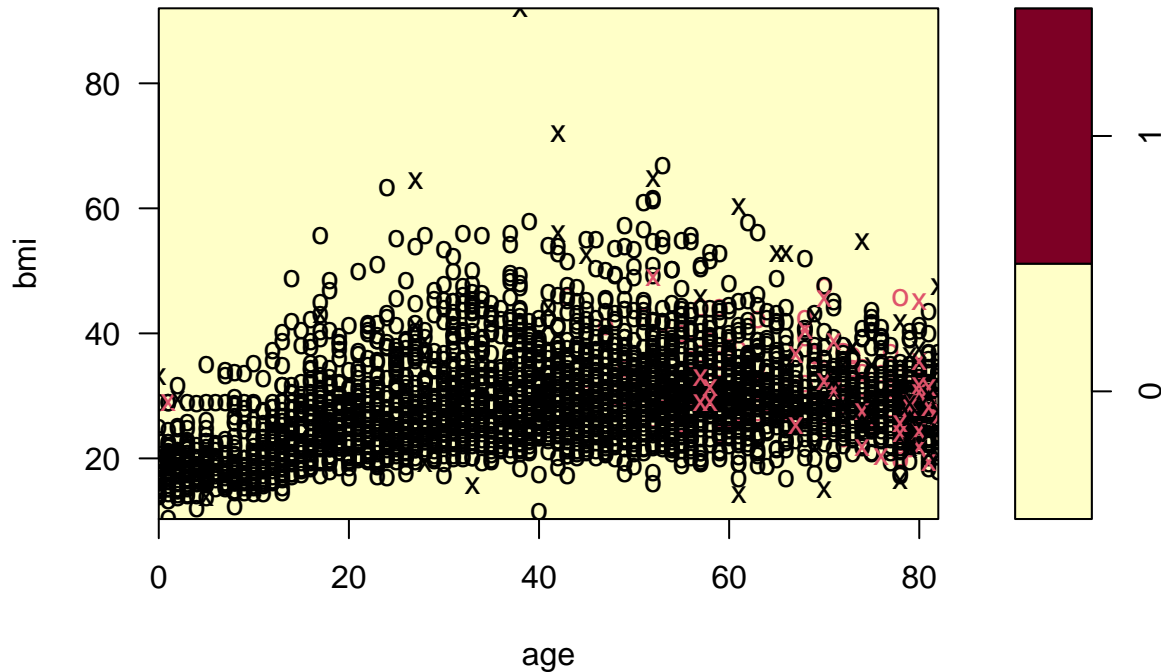
```
## 15      0.3      8      0      0
## 16      0.4      8      0      0
## 17      0.5      8      0      0
## 18      0.6      8      0      0
## 19      0.7      8      0      0
## 20      0.8      8      0      0
## 21      0.9      8      0      0
## 22      1.0      8      0      0
## 23      0.0     16      0      0
## 24      0.1     16      0      0
## 25      0.2     16      0      0
## 26      0.3     16      0      0
## 27      0.4     16      0      0
## 28      0.5     16      0      0
## 29      0.6     16      0      0
## 30      0.7     16      0      0
## 31      0.8     16      0      0
## 32      0.9     16      0      0
## 33      1.0     16      0      0
## 34      0.0     32      0      0
## 35      0.1     32      0      0
## 36      0.2     32      0      0
## 37      0.3     32      0      0
## 38      0.4     32      0      0
## 39      0.5     32      0      0
## 40      0.6     32      0      0
## 41      0.7     32      0      0
## 42      0.8     32      0      0
## 43      0.9     32      0      0
## 44      1.0     32      0      0
```

```
svm_after_tuned <- tuned_svm$best.model
summary(svm_after_tuned)
```

```
##
## Call:
## best.tune(method = svm, train.x = stroke ~ ., data = df_train_svm,
##   ranges = list(epsilon = seq(0, 1, 0.1), cost = 2^(2:5)))
##
##
## Parameters:
##   SVM-Type:  C-classification
##   SVM-Kernel: radial
##   cost:      4
##
## Number of Support Vectors: 95
##
##   ( 31 64 )
##
##
## Number of Classes: 2
##
## Levels:
## 0 1
```

```
plot(svm_after_tuned, data = df_train_svm, bmi ~ age, slice = list(avg_glucose_level = 3))
```

SVM classification plot



```
# Confusion matrix after tuning hyperparameters
svm_prediction_tuned <- predict(svm_after_tuned, newdata = df_test_svm)
svm_tuned_test_error <- mean(svm_prediction_tuned != ytest)
confusionMatrix(svm_prediction_tuned, df_test_svm$stroke)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##           0 1466    0
##           1    0    67
##
##           Accuracy : 1
##           95% CI : (0.9976, 1)
##           No Information Rate : 0.9563
##           P-Value [Acc > NIR] : < 2.2e-16
##
##           Kappa : 1
##
##           McNemar's Test P-Value : NA
##
##           Sensitivity : 1.0000
##           Specificity : 1.0000
##           Pos Pred Value : 1.0000
##           Neg Pred Value : 1.0000
##           Prevalence : 0.9563
```

```
##          Detection Rate : 0.9563
##    Detection Prevalence : 0.9563
##      Balanced Accuracy : 1.0000
##
##      'Positive' Class : 0
##
```

```
svm_tuned_test_error
```

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

```
confusionMatrix(svm_prediction_tuned,df_test_svm$stroke)
```

```
## Confusion Matrix and Statistics
##
##          Reference
## Prediction    0    1
##          0 1466    0
##          1    0   67
##
##          Accuracy : 1
##          95% CI : (0.9976, 1)
##    No Information Rate : 0.9563
##    P-Value [Acc > NIR] : < 2.2e-16
##
##          Kappa : 1
##
##    Mcnemar's Test P-Value : NA
##
##          Sensitivity : 1.0000
##          Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 1.0000
##          Prevalence : 0.9563
##          Detection Rate : 0.9563
##    Detection Prevalence : 0.9563
##          Balanced Accuracy : 1.0000
##
##      'Positive' Class : 0
##
```

11 Compare Models

12 OPTIONAL solve an optimisation problem

13 Conclusion