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

Lucerne University of Applied Sciences and Arts

Table of Contents

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. Therefore, our aim is to prevent stoke in future with the help of existing data and machine learning models.

In the following project 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.

For the following work, different R packages have been used. The respective packages and their installation can be found in the original R Markdown file or the README.

!! 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 and the missing values also concern data from people who had a stoke, 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)
```

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())
```

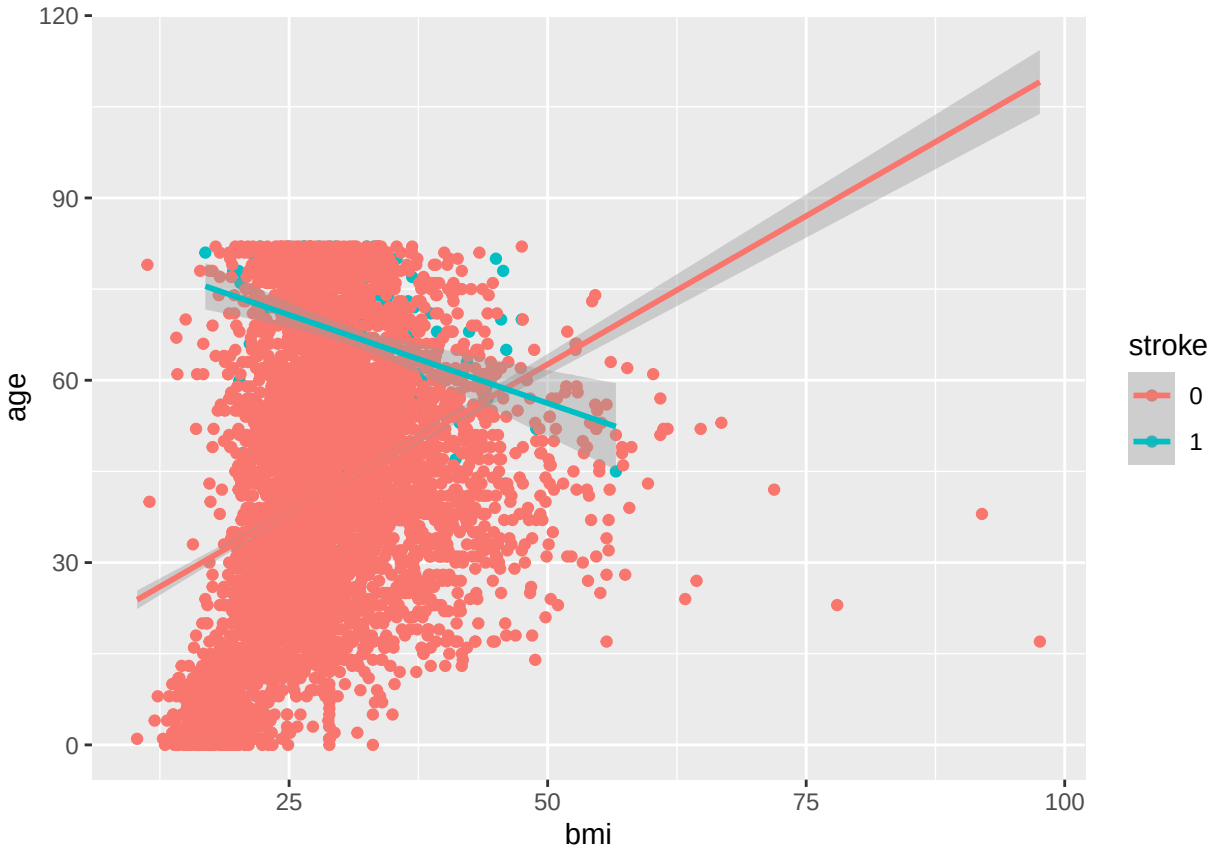
```
## `summarise()` has grouped output by 'ever_married'. You can override using the
## `.groups` argument.
```

3.2 Scatterplots

With the following Scatterplots the strength of the relation between the various parameters should be visualized in more detail. These leads to a better understanding of the data. For the sake of simplicity we do not include all plots, the whole code can be found in the rmarkdown file.

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

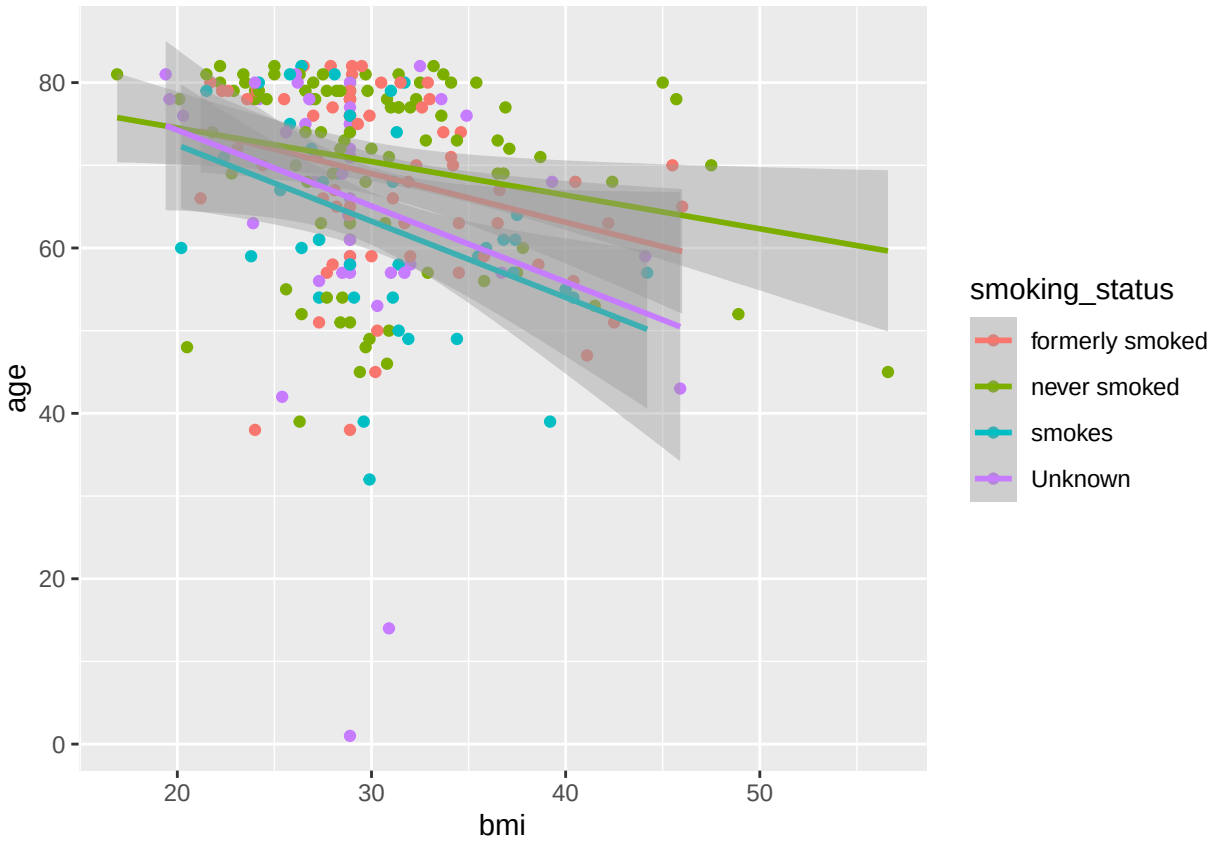
```
## `geom_smooth()` using formula 'y ~ x'
```




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

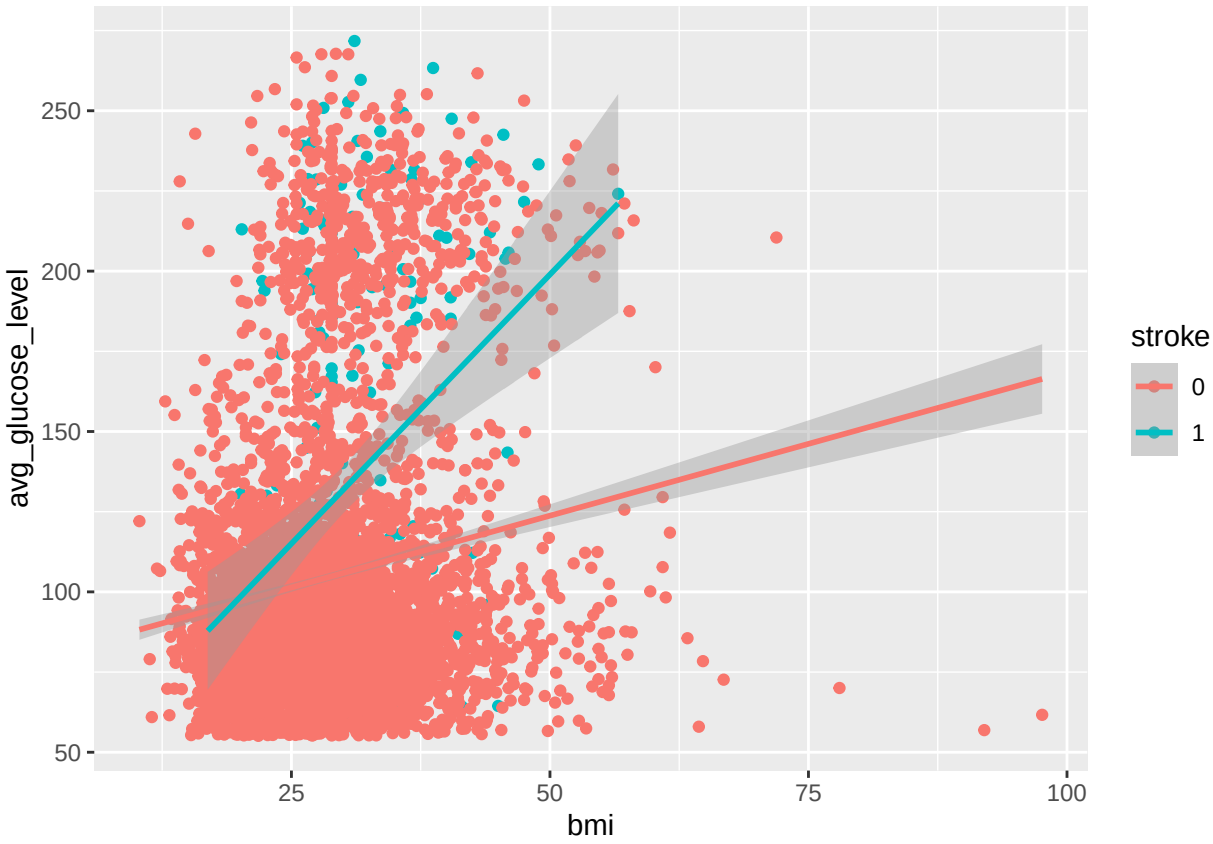
```
## Warning: Ignoring unknown parameters: method
```

```
## `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')
```

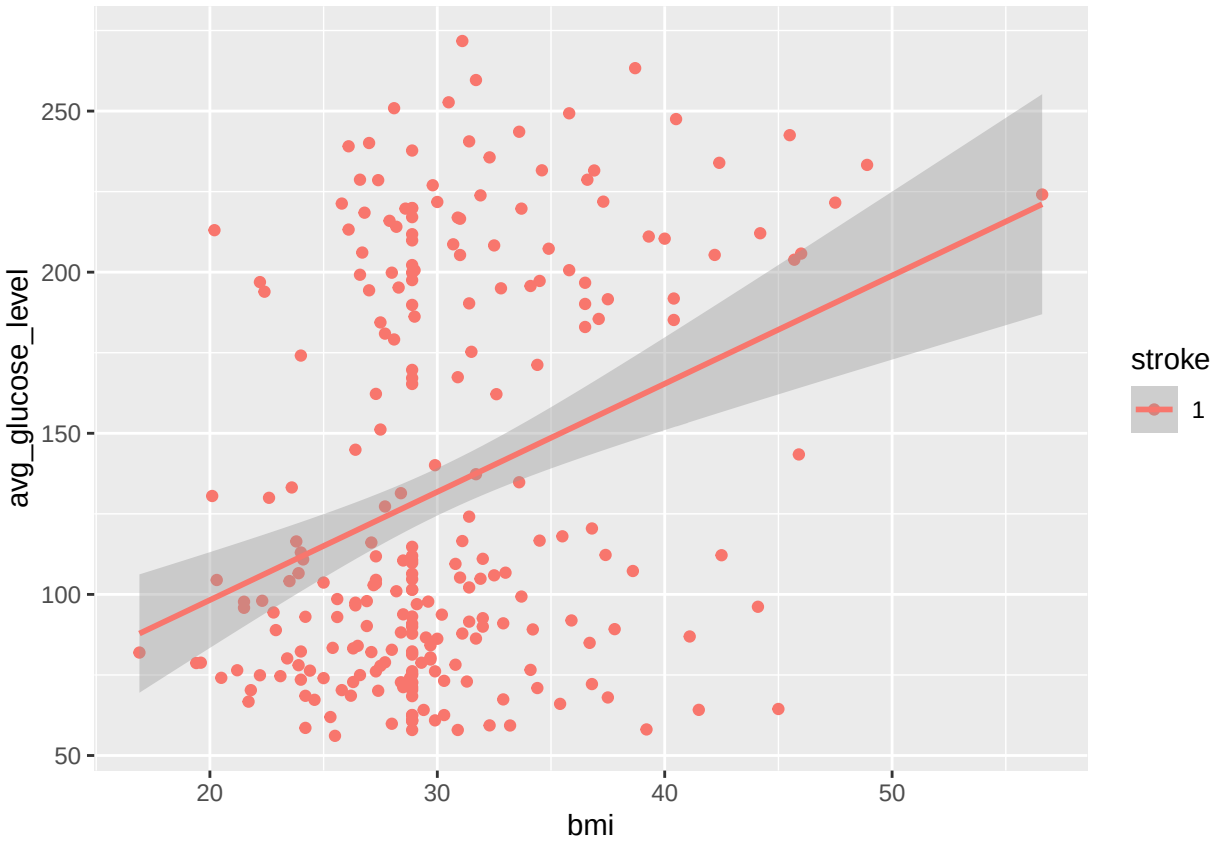
```
## `geom_smooth()` using formula 'y ~ x'
```




```
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
```

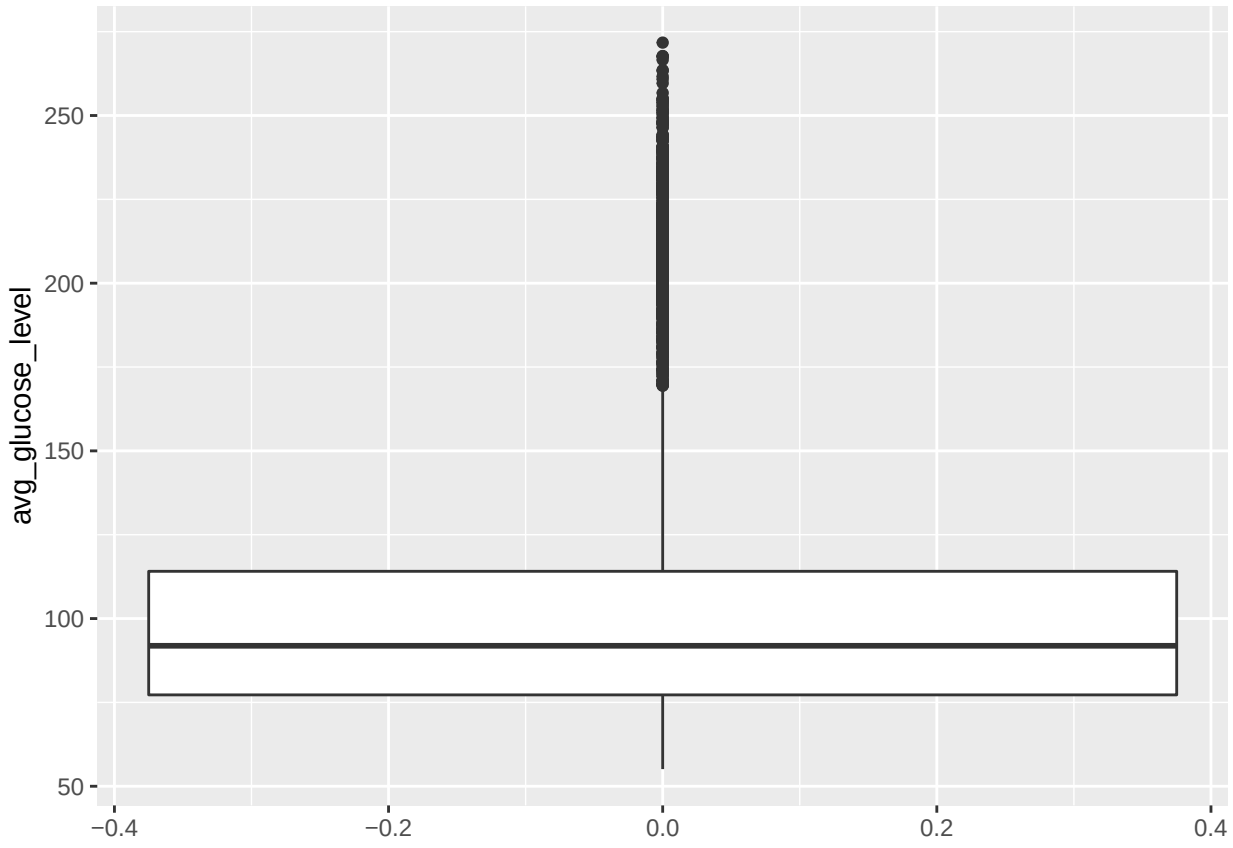
```
## `geom_smooth()` using formula 'y ~ x'
```



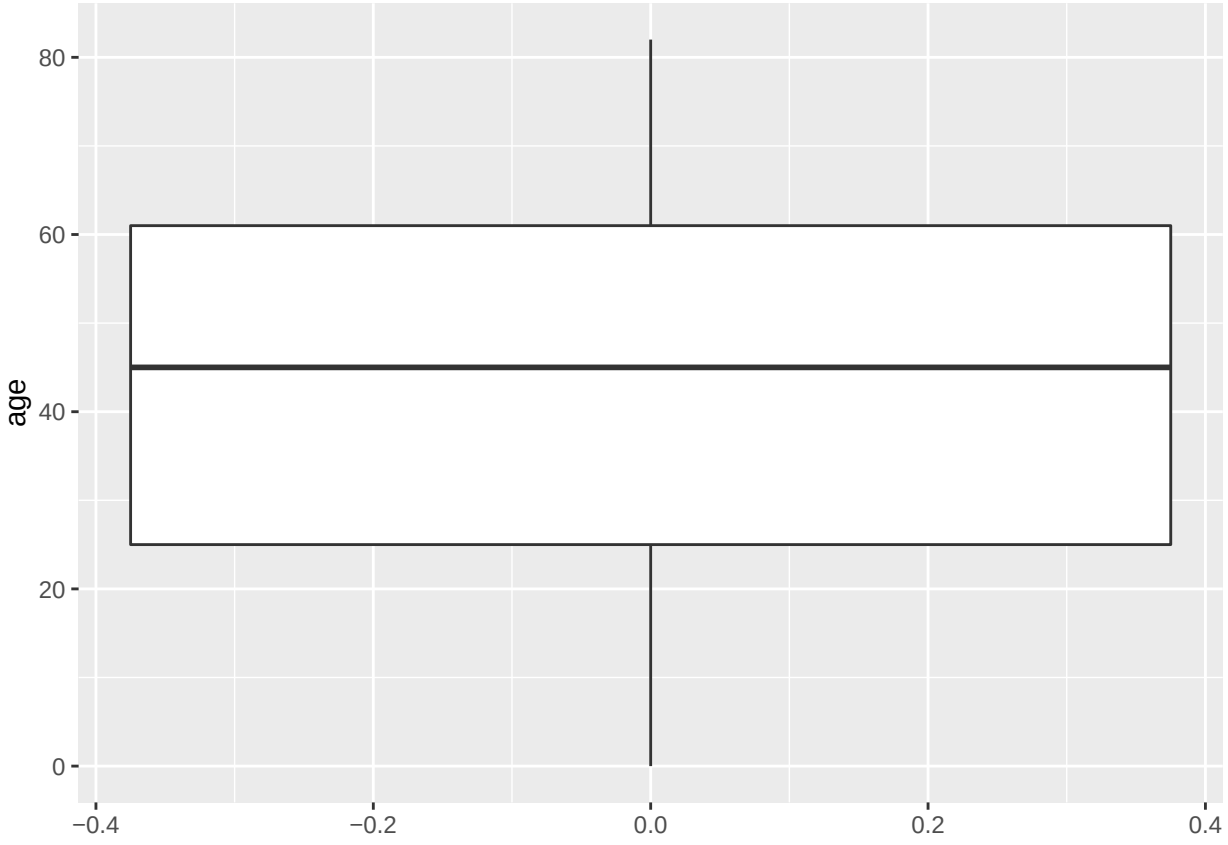
3.3 Boxplots

With the following Boxplots the distribution of the data points is visualized Again, not all Boxplots are included.

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



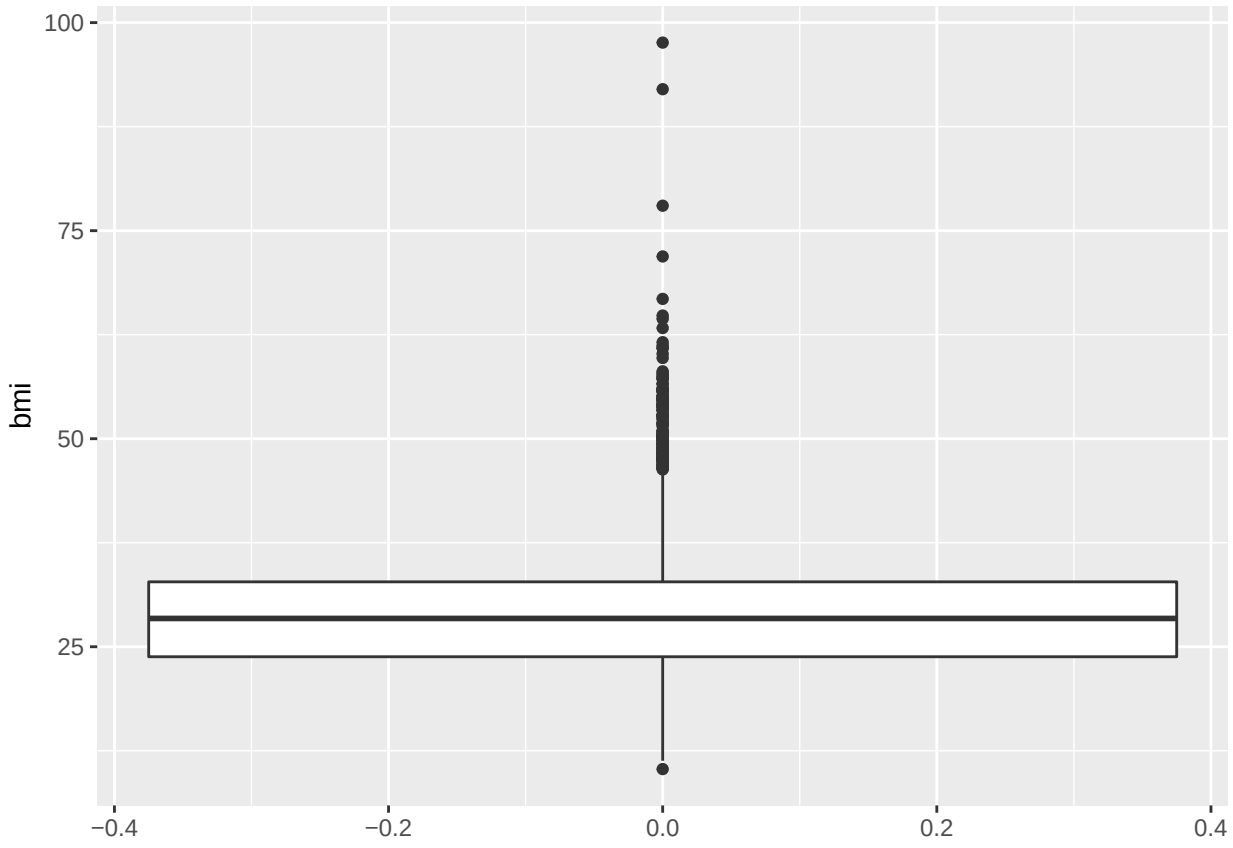

```
#Boxplot on Age
ggplot(stroke_data, aes(y = age)) +
  geom_boxplot()
```



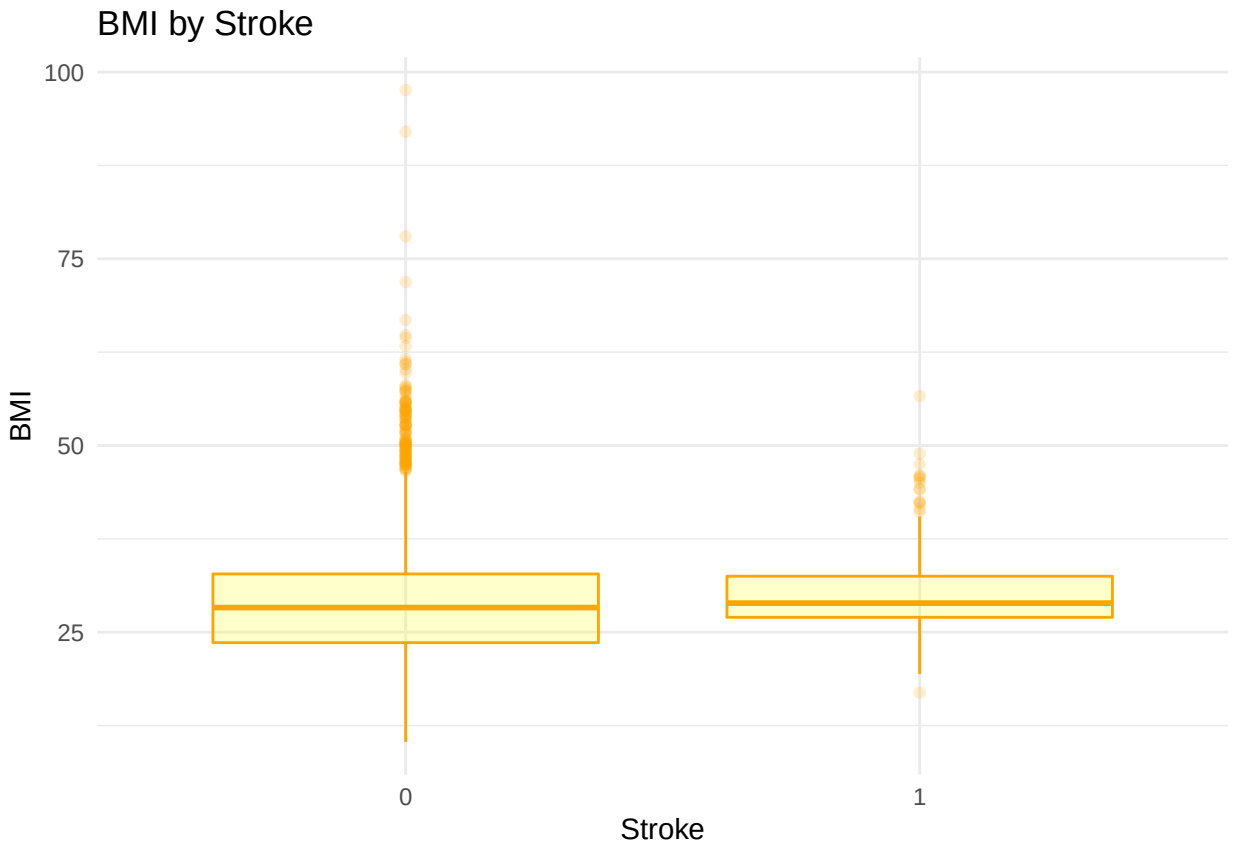

```
stroke_by_age <- stroke_data %>%
# dplyr::filter(stroke == 1) %>%
 ggplot(aes(x = stroke,
            y = age)) +
  geom_boxplot() +
  labs(title = "Boxplot on no-Stroke (0) and Stroke (1) by Age",
       x = "No-stroke (0) and Stroke (1)",
       y = "Age")
       color = "green"

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 str
# 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 We know of the following limitations for fitting a linear regression model to a binary response variable: - - ## Selecting Predictors for the model

Apart from testing out individual variables, we also tested a model with all predictors to find out about the significant ones, as selecting all variables may not necessarily lead to a more accurate model, and we might run into overfitting problems. Furthermore, our variable selection is also informed by our exploratory plots above, and what we intuitively believe to be accurate risk factors for a stroke.

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

```
## Warning in summary.lm(lm.stroke.test): essentially perfect fit: summary may be
## unreliable
```

```
##
## Call:
## lm(formula = stroke_num ~ ., data = df_train_linear)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.124e-15 -2.180e-18  4.000e-20  2.250e-18  9.299e-17
##
## Coefficients:
##                              Estimate Std. Error    t value Pr(>|t|)
## (Intercept)                -4.797e-18  2.196e-18 -2.184e+00 0.029032 *
## id                          1.498e-23  1.713e-23  8.740e-01 0.382064
## genderMale                  4.919e-19  7.542e-19  6.520e-01 0.514290
## genderOther                -3.276e-18  2.193e-17 -1.490e-01 0.881225
## age                         3.668e-19  2.793e-20  1.313e+01  < 2e-16 ***
## hypertension                4.109e-18  1.265e-18  3.247e+00 0.001176 **
## heart_disease               7.985e-18  1.674e-18  4.771e+00 1.91e-06 ***
## ever_marriedYes            -4.324e-18  1.093e-18 -3.954e+00 7.83e-05 ***
## work_typeGovt_job          -7.648e-18  1.941e-18 -3.941e+00 8.27e-05 ***
## work_typeNever_worked      -3.312e-18  5.611e-18 -5.900e-01 0.555056
## work_typePrivate           -5.647e-18  1.630e-18 -3.464e+00 0.000538 ***
```

```
## work_typeSelf-employed     -9.323e-18  1.981e-18 -4.705e+00 2.63e-06 ***
## residence_typeUrban         7.382e-20  7.327e-19  1.010e-01 0.919760
## avg_glucose_level           3.220e-20  8.396e-21  3.835e+00 0.000128 ***
## bmi                        -5.681e-20  5.429e-20 -1.046e+00 0.295421
## smoking_statusnever smoked -8.056e-19  1.086e-18 -7.420e-01 0.458274
## smoking_statussmokes       -6.396e-19  1.295e-18 -4.940e-01 0.621509
## smoking_statusUnknown      -1.744e-19  1.215e-18 -1.440e-01 0.885900
## stroke1                     1.000e+00  1.751e-18  5.713e+17  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.187e-17 on 3558 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 2.006e+34 on 18 and 3558 DF,  p-value: < 2.2e-16
```

Out of the significant predictors above, we chose the following model for the linear model, glm, and gam:

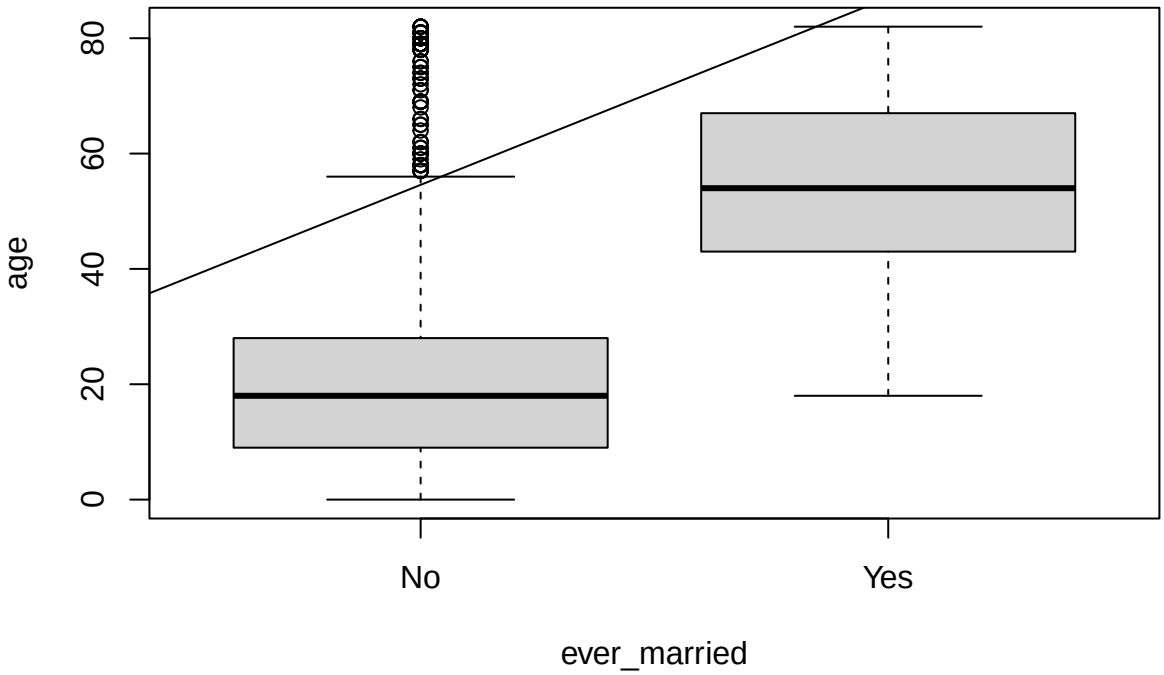
```
# fitting models for simple regression model
linear_model <- stroke_num ~ age + hypertension + heart_disease + avg_glucose_level
```

Although ever_married theoretically returns a significant result, this is most probably confounded by age, as can be seen in the plot below, the subjects that are or were married at some point tend to be significantly older than those that have never married. The same is most likely true for the work type, with those having never worked being younger. What surprised us, though, is that smoking status did not have as big of an effect as we would have anticipated.

```
# fitting models for simple regression model
lm.married <- lm(age ~ ever_married, data = df_train_linear)
summary(lm.married)
```

```
##
## Call:
## lm(formula = age ~ ever_married, data = df_train_linear)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -36.575 -12.575  -1.575  10.425  59.838
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)      22.1622     0.4849   45.70   <2e-16 ***
## ever_marriedYes  32.4132     0.5944   54.53   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 16.77 on 3575 degrees of freedom
## Multiple R-squared:  0.4541, Adjusted R-squared:  0.454
## F-statistic:  2974 on 1 and 3575 DF,  p-value: < 2.2e-16
```

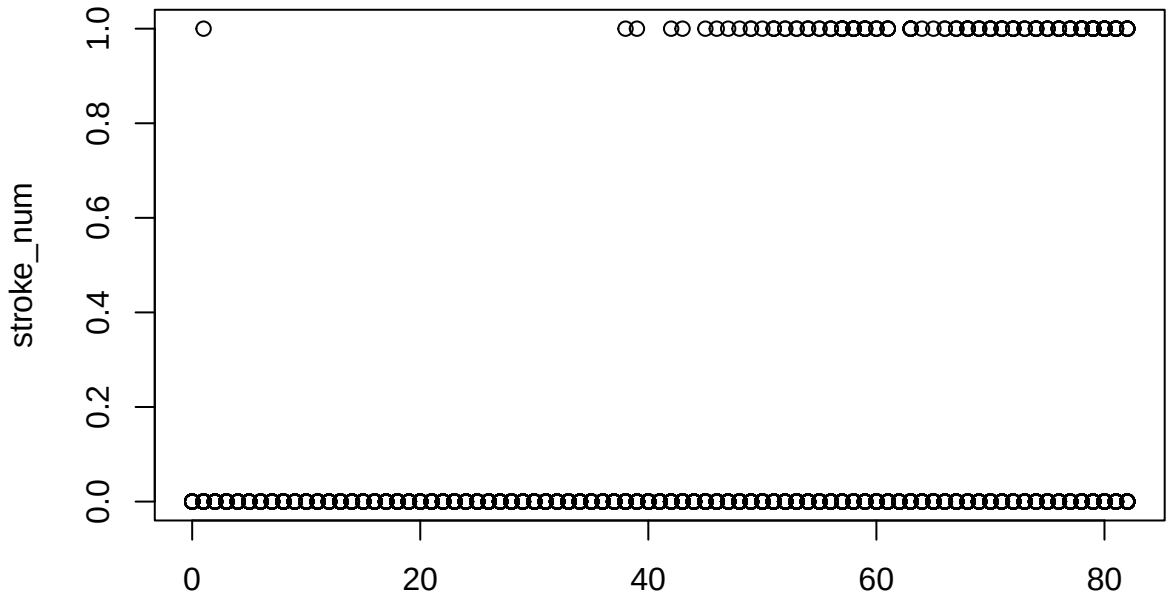
```
plot(age ~ ever_married, data = df_train_linear)
abline(lm.married)
```

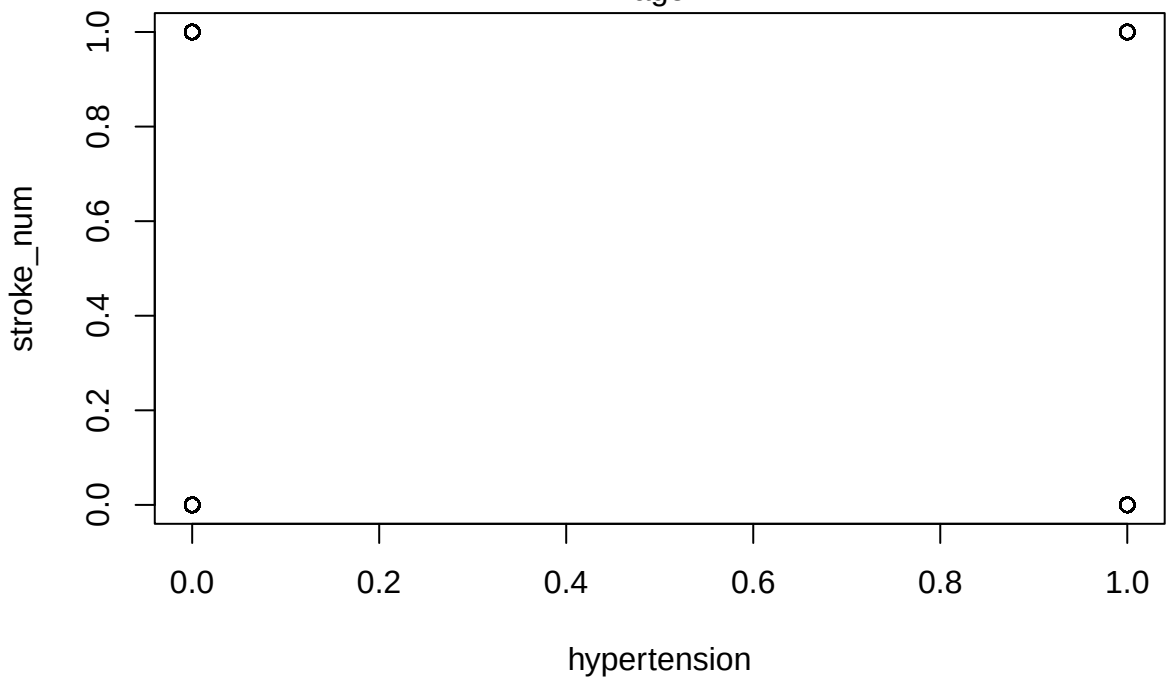


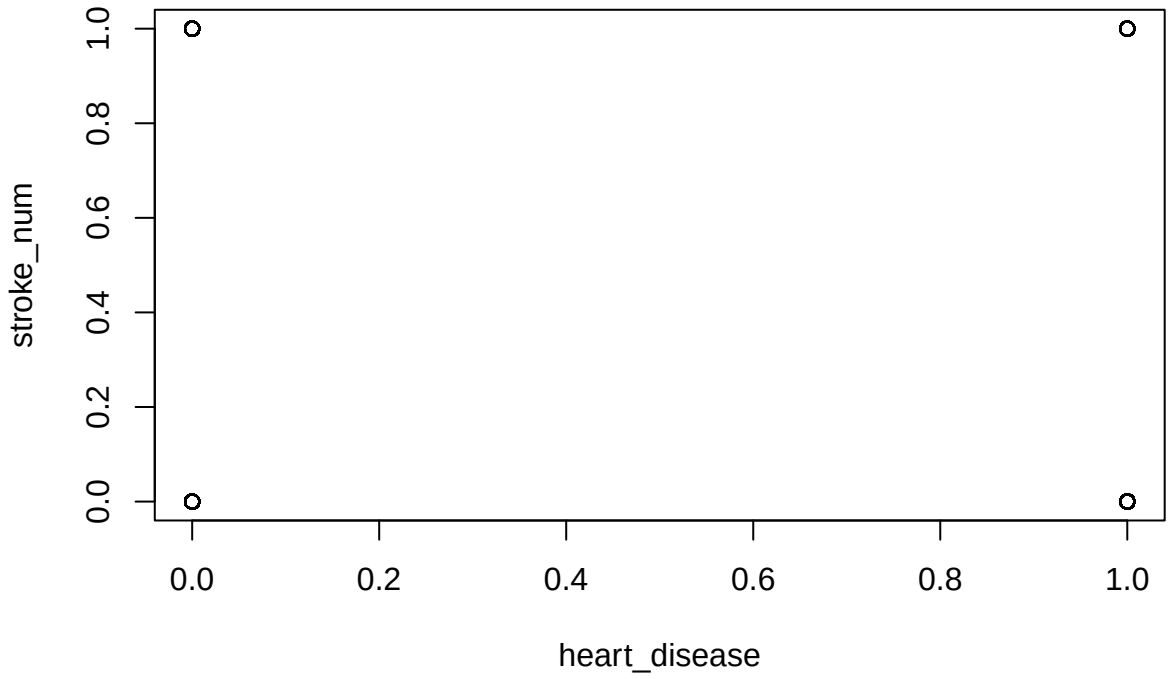

```
# fitting models for simple regression model
lm.stroke <- lm(stroke_num ~ age + hypertension + heart_disease + avg_glucose_level, data = df_train_li
summary(lm.stroke)
```

```
##
## Call:
## lm(formula = stroke_num ~ age + hypertension + heart_disease +
##     avg_glucose_level, data = df_train_linear)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.27445 -0.07958 -0.03644  0.00587  1.05238
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -7.611e-02  1.050e-02  -7.246 5.24e-13 ***
## age                1.939e-03  1.696e-04  11.433  < 2e-16 ***
## hypertension       3.874e-02  1.205e-02   3.214 0.001319 **
## heart_disease      8.643e-02  1.587e-02   5.444 5.55e-08 ***
## avg_glucose_level  3.097e-04  7.986e-05   3.877 0.000108 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2107 on 3572 degrees of freedom
## Multiple R-squared:  0.08211,    Adjusted R-squared:  0.08108
## F-statistic: 79.88 on 4 and 3572 DF,  p-value: < 2.2e-16
```

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

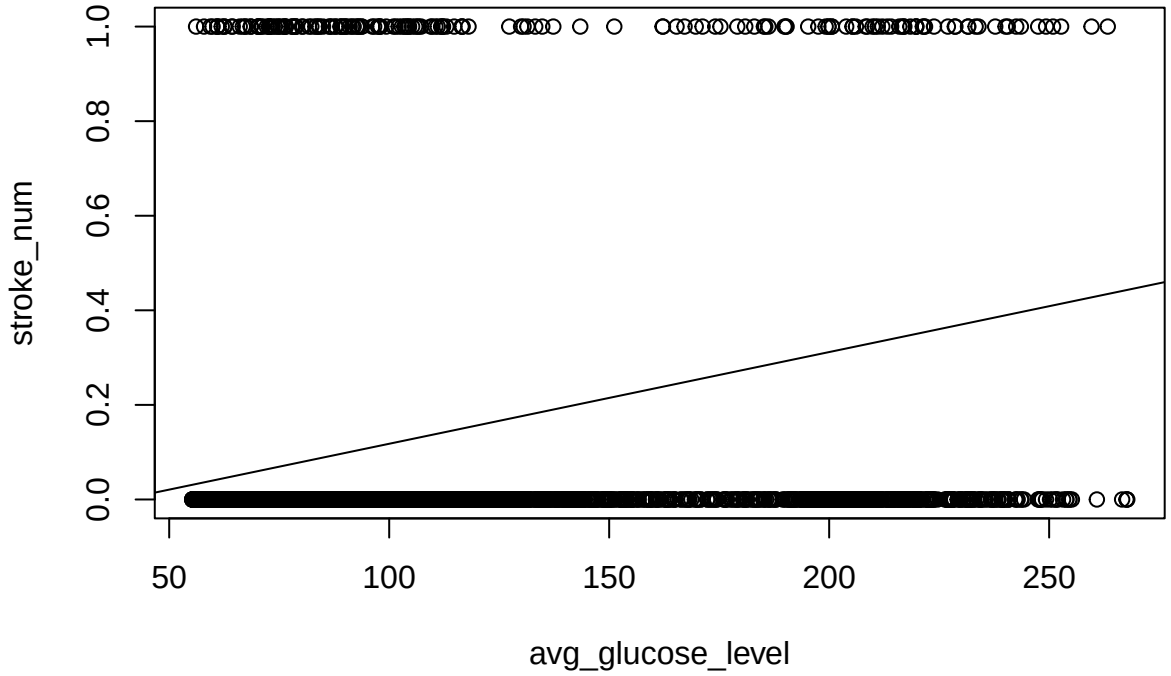

stroke_num
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80
age


stroke_num
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0
hypertension




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

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



Next, we evaluate the model fit with a confusion matrix. In the first example, we have constructed one "manually". In all further examples we will be using the caret package's confusionMatrix() function.

```
# Evaluating model fit using predict linear regression, example of a self-constructed confusion matrix
```

```
predicted.lm.stroke <- ifelse(predict(lm.stroke, df_test_linear, type = "response") < 0.15, yes = 0, no
head(predicted.lm.stroke)
```

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

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

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

```
##    fit
## obs    0    1
##   0 1390   76
##   1   56   11
```

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

```
##    fit
## obs    0    1
##   0 0.91 0.05
##   1 0.04 0.01
```

```
# Evaluating model fit using predict linear regression using confusionMatrix()

predicted.lm.stroke <- ifelse(predict(lm.stroke, df_test_linear, type = "response") < 0.15, yes = 0, no
head(predicted.lm.stroke)
```

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

```
confusionMatrix(as.factor(predicted.lm.stroke), as.factor(df_test_linear$stroke))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1390   56
##          1   76   11
##
##                Accuracy : 0.9139
##                  95% CI : (0.8987, 0.9275)
##     No Information Rate : 0.9563
##     P-Value [Acc > NIR] : 1.00000
##
##                   Kappa : 0.0983
##
##  Mcnemar's Test P-Value : 0.09818
##
```

```
##             Sensitivity : 0.9482
##             Specificity : 0.1642
##          Pos Pred Value : 0.9613
##          Neg Pred Value : 0.1264
##              Prevalence : 0.9563
##          Detection Rate : 0.9067
##    Detection Prevalence : 0.9432
##       Balanced Accuracy : 0.5562
##
##        'Positive' Class : 0
##
```

6 Generalised Linear Model with family set to Poisson

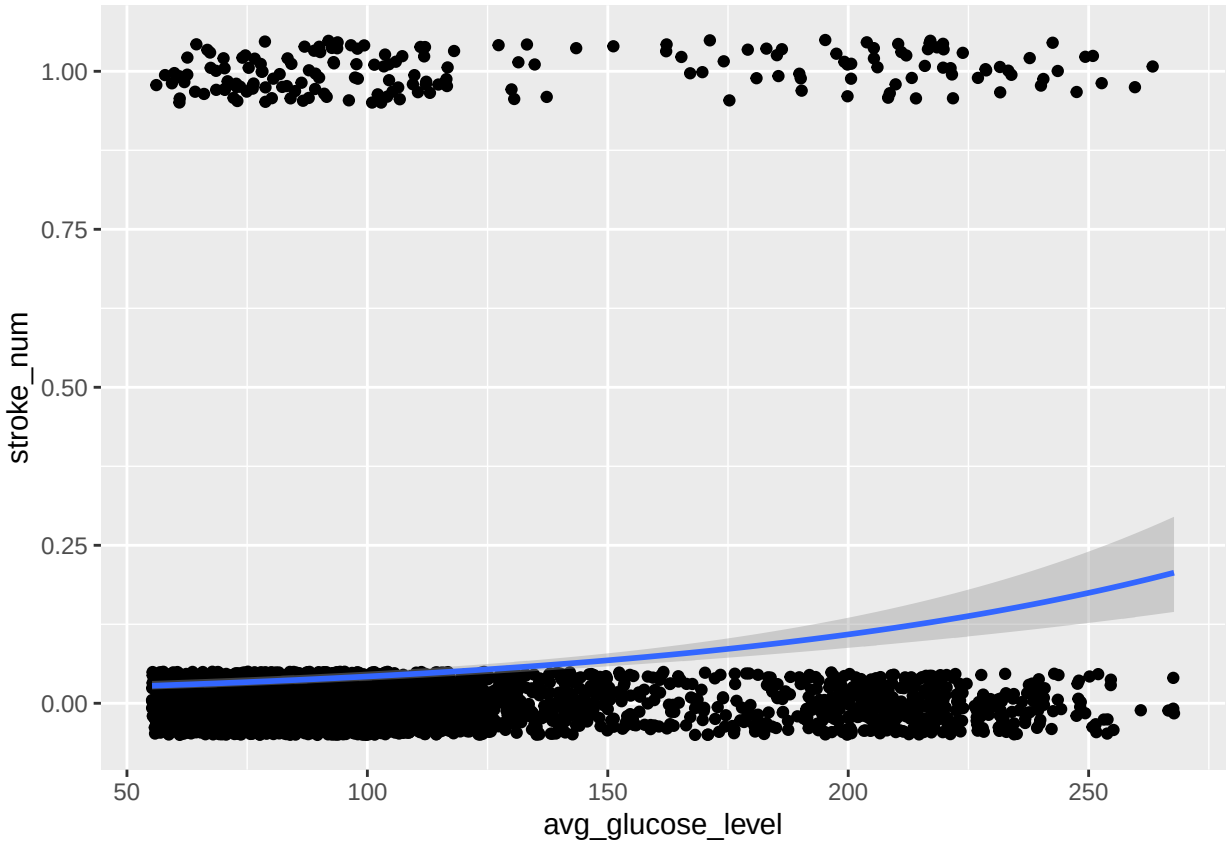
```
# text
glm.stroke.poisson <- glm(stroke_num ~ age + heart_disease + hypertension + avg_glucose_level,
family = "poisson",
data = df_train_linear)
summary(glm.stroke.poisson)
```

```
##
## 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'
```



Below is comparison of the fitted values versus the actual observed values within the training data set.

```
# Evaluating model fit poisson

# fitted(glm.stroke.poisson)
fitted.glm.stroke.poisson <- ifelse(fitted(glm.stroke.poisson) < 0.2, yes = 0, no = 1)
head(fitted.glm.stroke.poisson)
```

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

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

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

```
##    fit
## obs    0    1
##   0 3238  157
##   1  130   52
```

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

```
##    fit
## obs    0    1
##   0 0.91 0.04
##   1 0.04 0.01
```

```
# Evaluating model fit using predict linear regression using confusionMatrix()

glm.stroke.poisson.prediction <- as.factor(ifelse(predict(glm.stroke.poisson, df_test_linear, type = "r
head(glm.stroke.poisson.prediction)
```

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

```
confusionMatrix(glm.stroke.poisson.prediction, df_test_linear$stroke)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1355   46
##          1  111   21
##
##                Accuracy : 0.8976
##                  95% CI : (0.8813, 0.9123)
##     No Information Rate : 0.9563
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1625
##
##  Mcnemar's Test P-Value : 3.26e-07
##
##             Sensitivity : 0.9243
##             Specificity : 0.3134
##          Pos Pred Value : 0.9672
##          Neg Pred Value : 0.1591
##              Prevalence : 0.9563
##          Detection Rate : 0.8839
##    Detection Prevalence : 0.9139
##       Balanced Accuracy : 0.6189
##
##        'Positive' Class : 0
##
```

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 ~ age + gender + avg_glucose_level + residence_type + work_type + hea
family = "binomial",
data = df_train_linear)
summary(glm.stroke.binomial)
```

```
##
## Call:
## glm(formula = stroke ~ age + gender + avg_glucose_level + residence_type +
##     work_type + heart_disease + hypertension + ever_married +
##     bmi + smoking_status, family = "binomial", data = df_train_linear)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.3023  -0.3211  -0.1577  -0.0778   3.6948
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                 -7.320e+00  1.079e+00  -6.781 1.19e-11 ***
## age                          8.025e-02  7.152e-03  11.221  < 2e-16 ***
## genderMale                   1.941e-01  1.666e-01   1.165  0.24399
## genderOther                 -1.029e+01  1.455e+03  -0.007  0.99436
## avg_glucose_level            3.678e-03  1.397e-03   2.632  0.00848 **
## residence_typeUrban         -2.994e-02  1.634e-01  -0.183  0.85465
## work_typeGovt_job           -9.598e-01  1.139e+00  -0.843  0.39939
## work_typeNever_worked       -1.015e+01  3.623e+02  -0.028  0.97766
## work_typePrivate            -7.741e-01  1.122e+00  -0.690  0.49033
## work_typeSelf-employed      -1.335e+00  1.145e+00  -1.166  0.24356
## heart_disease                4.014e-01  2.150e-01   1.867  0.06191 .
## hypertension                 3.709e-01  1.903e-01   1.949  0.05132 .
## ever_marriedYes             -2.280e-01  2.667e-01  -0.855  0.39271
## bmi                          8.944e-03  1.338e-02   0.669  0.50380
## smoking_statusnever smoked -1.636e-01  2.064e-01  -0.793  0.42796
## smoking_statussmokes         4.783e-02  2.569e-01   0.186  0.85228
## smoking_statusUnknown       -7.281e-02  2.442e-01  -0.298  0.76563
## ---
## 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: 1115.1  on 3560  degrees of freedom
## AIC: 1149.1
##
## Number of Fisher Scoring iterations: 14
```

```
prediction.glm <- predict(glm.stroke.binomial, df_test_linear, type = "response")
# InformationValue::optimalCutoff(df_test_linear, prediction.glm)
```

```
predicted.glm.stroke.binomial <- as.factor(ifelse(predict(glm.stroke.binomial, df_test_linear, type = "
head(predicted.glm.stroke.binomial)
```

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

```
confusionMatrix(predicted.glm.stroke.binomial, df_test_linear$stroke)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1351   44
##          1  115   23
##
##                Accuracy : 0.8963
##                  95% CI : (0.8799, 0.9111)
##     No Information Rate : 0.9563
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1759
##
##  Mcnemar's Test P-Value : 2.835e-08
##
##             Sensitivity : 0.9216
##             Specificity : 0.3433
##          Pos Pred Value : 0.9685
##          Neg Pred Value : 0.1667
##              Prevalence : 0.9563
##          Detection Rate : 0.8813
##    Detection Prevalence : 0.9100
##       Balanced Accuracy : 0.6324
##
##        'Positive' Class : 0
##
```

```
# First iteration with parameters chosen from intuitive domain knowledge and exploratory analysis of da
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, dat
#abline(glm.stroke.binomial)
```

```
# 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
```

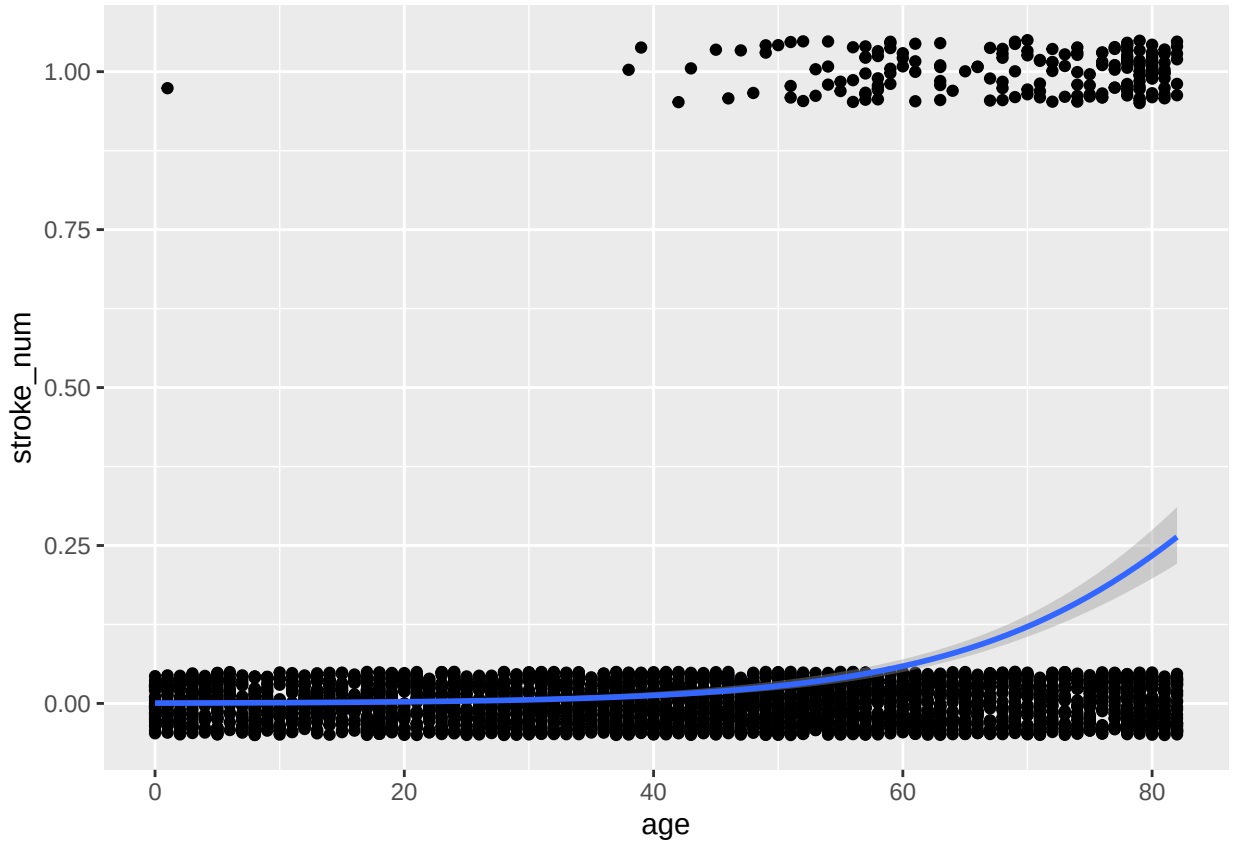
```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1438.7  on 3576  degrees of freedom
## Residual deviance: 1127.6  on 3572  degrees of freedom
## AIC: 1137.6
##
## Number of Fisher Scoring iterations: 7
```

```
# plot(stroke ~ age + heart_disease + hypertension + avg_glucose_level, data = df_train_linear)
# abline(glm.stroke.binomial.2)

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'
```




```
predicted.glm.stroke.binomial.2 <- as.factor(ifelse(predict(glm.stroke.binomial.2, df_test_linear, type
head(predicted.glm.stroke.binomial.2)
```

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

```
## 0 0 0 0 0 0
## Levels: 0 1
```

```
# predicted.glm.stroke.binomial.2 <- as.factor(predicted.glm.stroke.binomial.2)
# predicted.glm.stroke.binomial.2 <- relevel(predicted.glm.stroke.binomial.2, "0")

confusionMatrix(predicted.glm.stroke.binomial.2, df_test_linear$stroke)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1412   54
##          1   54   13
##
##                Accuracy : 0.9295
##                  95% CI : (0.9156, 0.9419)
##     No Information Rate : 0.9563
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1572
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9632
##             Specificity : 0.1940
##          Pos Pred Value : 0.9632
##          Neg Pred Value : 0.1940
##              Prevalence : 0.9563
##          Detection Rate : 0.9211
##    Detection Prevalence : 0.9563
##       Balanced Accuracy : 0.5786
##
##        'Positive' Class : 0
##
```

8 Generalised Additive Model

```
library(mgcv)

gam.stroke <- gam(stroke_num ~ age + heart_disease + hypertension + avg_glucose_level,
family = "binomial",
data = df_train_linear)
summary(gam.stroke)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
```

```
## stroke_num ~ age + heart_disease + hypertension + avg_glucose_level
##
## Parametric coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)       -7.739236   0.443974 -17.432   <2e-16 ***
## age                0.072374   0.006312  11.466   <2e-16 ***
## heart_disease      0.465121   0.211313   2.201   0.0277 *
## hypertension       0.362843   0.187105   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
##
##
## R-sq.(adj) =  0.101   Deviance explained = 21.6%
## UBRE = -0.68197  Scale est. = 1         n = 3577
```

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

```
predicted.gam.stroke <- as.factor(ifelse(predict( gam.stroke, df_test_linear, type = "response") < 0.2,
head(predicted.gam.stroke)
```

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

```
confusionMatrix(predicted.gam.stroke, df_test_linear$stroke)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1412   54
##          1   54   13
##
##                Accuracy : 0.9295
##                  95% CI : (0.9156, 0.9419)
##     No Information Rate : 0.9563
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1572
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9632
##             Specificity : 0.1940
##          Pos Pred Value : 0.9632
##          Neg Pred Value : 0.1940
##              Prevalence : 0.9563
##          Detection Rate : 0.9211
##    Detection Prevalence : 0.9563
##       Balanced Accuracy : 0.5786
```

```
##
##        'Positive' Class : 0
##
```

9 Neural Network Yves

10 Support Vector Machine (Larissa)

Stroke Data Classification using a Support Vector Machine. In the following two different approaches for the Stroke Data Classification are used. On one hand to compare the different approaches and on the other hand for the learning purpose.

In the following model "svm_linear" the SVM model is implemented with the caret package. Our target variable is the stroke parameter which contains 0 (no-stroke) and 1 (stroke) which we want to predict. The first step is to split the data set into training and testing data. The training data is used for building the model and the testing data for evaluating the model.

```
#split the data into test and train data with a split of 70 (training) / 30 (testing)
set.seed(7406)
intrain <- createDataPartition(y = stroke_data$stroke, p = 0.7, list = FALSE)   #selecting our target v
training <- stroke_data[intrain, ]
testing <- stroke_data[-intrain, ]
```

```
#checking the dimension of the training and testing data set
dim(training)
```

```
## [1] 3578   13
```

```
dim(testing)
```

```
## [1] 1532   13
```

Implement the trainControl Method. This function will control the computation overheads which allow us to use the train function which is implemented by the caret package. In the following we use the trainControl function with the repeated cross-validation method, with a iteration of 10 and with repeates set to 3 to compute the repeated cross-validation 3 times.

```
#implement the train control method with the repeated cross-validation method,
trctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
```

In the following we pass the trainControl function to our training method. In the train function we pass our target variable stroke and set the method to linear. The following output shows our train method. As our model was trained with the C value as one we now can predict our testing data set.

```
#implement the train function from caret
svm_linear <- train(stroke ~., data = training, method = "svmLinear",
                    trControl=trctrl,
                    prepProcess = c("center", "scale"),
                    tuneLenght = 10)
svm_linear
```

```
## Support Vector Machines with Linear Kernel
##
## 3578 samples
##   12 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 3220, 3219, 3221, 3220, 3221, 3220, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9178854  0.09343552
##
## Tuning parameter 'C' was held constant at a value of 1
```

When running our testing model we will get the prediction values with 0 (no-stroke) and 1 (stroke).

```
#we use the predict function and pass the trained model
#and as new data we pass the testing data set
test_pred <- predict(svm_linear, newdata = testing)
test_pred
```

```
##    [1] 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0
##   [38] 1 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
##   [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [112] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [149] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [186] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [223] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [260] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [297] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [334] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
##  [371] 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [408] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [445] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [482] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [519] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [556] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [593] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [630] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [667] 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [704] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [741] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [778] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [815] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [852] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [889] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
##  [926] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [963] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1000] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1037] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1074] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
## [1111] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1148] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1185] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1222] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1259] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1296] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1333] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1370] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1407] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1444] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
## [1481] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1518] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## Levels: 0 1
```

In the next step we are going to test the accuracy of our model. We do this with the help of the confusionMatrix. The following output shows, that our model has an accuracy of 95%. Which seems to be quite good. However for the sake of completness, we going to improve our model with customizing cost values.

```
#testing the accuracy of our model with confusionMatrix
confusionMatrix(table(test_pred, testing$stroke))
```

```
## Confusion Matrix and Statistics
##
##
## test_pred    0    1
##         0 1448   64
##         1   10   10
##
##                Accuracy : 0.9517
##                  95% CI : (0.9397, 0.9619)
##     No Information Rate : 0.9517
##     P-Value [Acc > NIR] : 0.5309
##
##                   Kappa : 0.1962
##
##  Mcnemar's Test P-Value : 7.223e-10
##
##             Sensitivity : 0.9931
##             Specificity : 0.1351
##          Pos Pred Value : 0.9577
##          Neg Pred Value : 0.5000
##              Prevalence : 0.9517
##          Detection Rate : 0.9452
##    Detection Prevalence : 0.9869
##       Balanced Accuracy : 0.5641
##
##        'Positive' Class : 0
##
```

```
#define different values for c
grid <- expand.grid(C = c(0, 0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2.5))
```

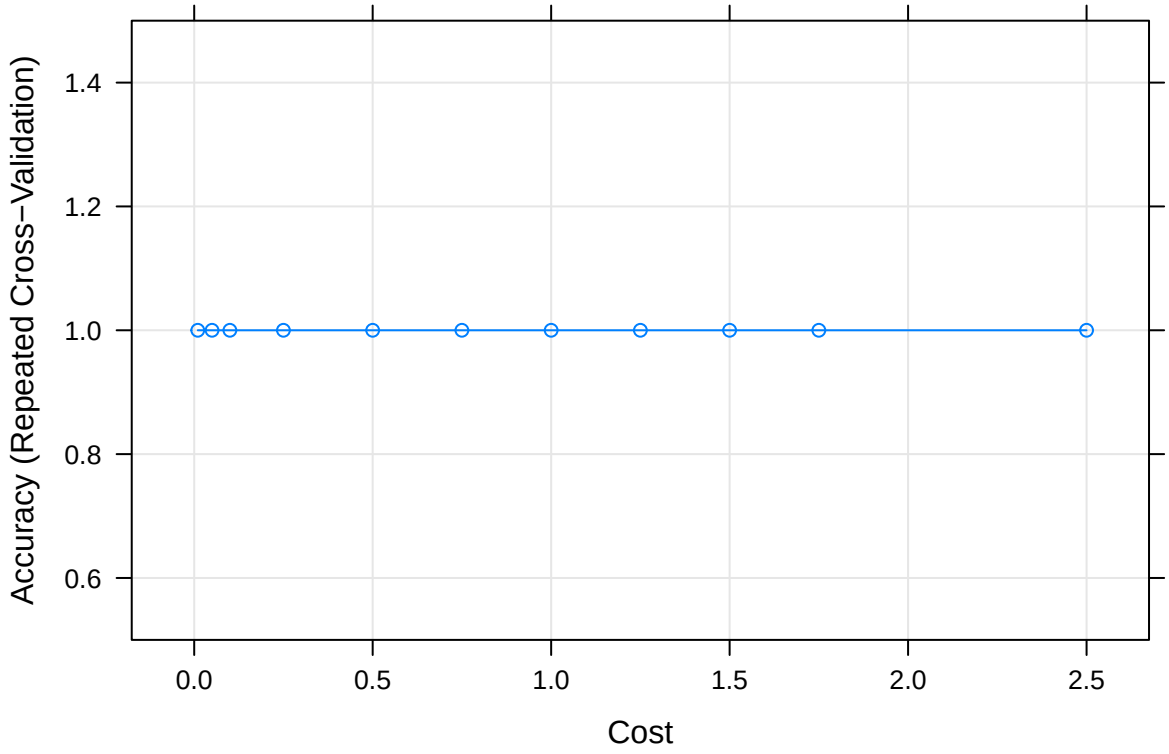
The following output shows that the model shows the final value used for the model was C = 0.01.

```r
#improving the performance
svm_linear_grid <- train(stroke ~., data = training, method = "svmLinear",
                         trControl=trctrl,
                         preProcess = c("center", "scale"),
                         tuneGrid = grid,
                         tuneLength = 10)

svm_linear_grid
```

```
## Support Vector Machines with Linear Kernel
##
## 3578 samples
##   12 predictor
##    2 classes: '0', '1'
##
## Pre-processing: centered (18), scaled (18)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 3221, 3221, 3221, 3220, 3221, 3219, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy  Kappa
##   0.00  NaN       NaN
##   0.01    1         1
##   0.05    1         1
##   0.10    1         1
##   0.25    1         1
##   0.50    1         1
##   0.75    1         1
##   1.00    1         1
##   1.25    1         1
##   1.50    1         1
##   1.75    1         1
##   2.50    1         1
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 0.01.
```

```r
plot(svm_linear_grid)
```




```
test_pred_grid <- predict(svm_linear_grid, newdata = testing)
test_pred_grid
```

```
##    [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##   [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##   [75] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [112] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [149] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [186] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [223] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [260] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [297] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [334] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [371] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [408] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [445] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [482] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [519] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [556] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [593] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [630] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [667] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [704] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [741] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [778] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [815] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [852] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [889] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [926] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
##  [963] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1000] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1037] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1074] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1111] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1148] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1185] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1222] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1259] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1296] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1333] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1370] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1407] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1444] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1481] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [1518] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## Levels: 0 1
```

```
confusionMatrix(table(test_pred_grid, testing$stroke))
```

```
## Confusion Matrix and Statistics
##
##
## test_pred_grid    0    1
##              0 1458    0
##              1    0   74
##
##                Accuracy : 1
##                  95% CI : (0.9976, 1)
##     No Information Rate : 0.9517
##     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 : 0.9517
##          Detection Rate : 0.9517
##    Detection Prevalence : 0.9517
##       Balanced Accuracy : 1.0000
##
##        'Positive' Class : 0
##
```

In the following the second approach for the SVM Model is presented. In a first step we again split the data set into training and testing data sets.

```r
#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
```

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

```r
#define stroke train and test variables
ytrain = df_train_svm$stroke
ytest = df_test_svm$stroke
```

```r
table(df_train_svm$stroke)
```

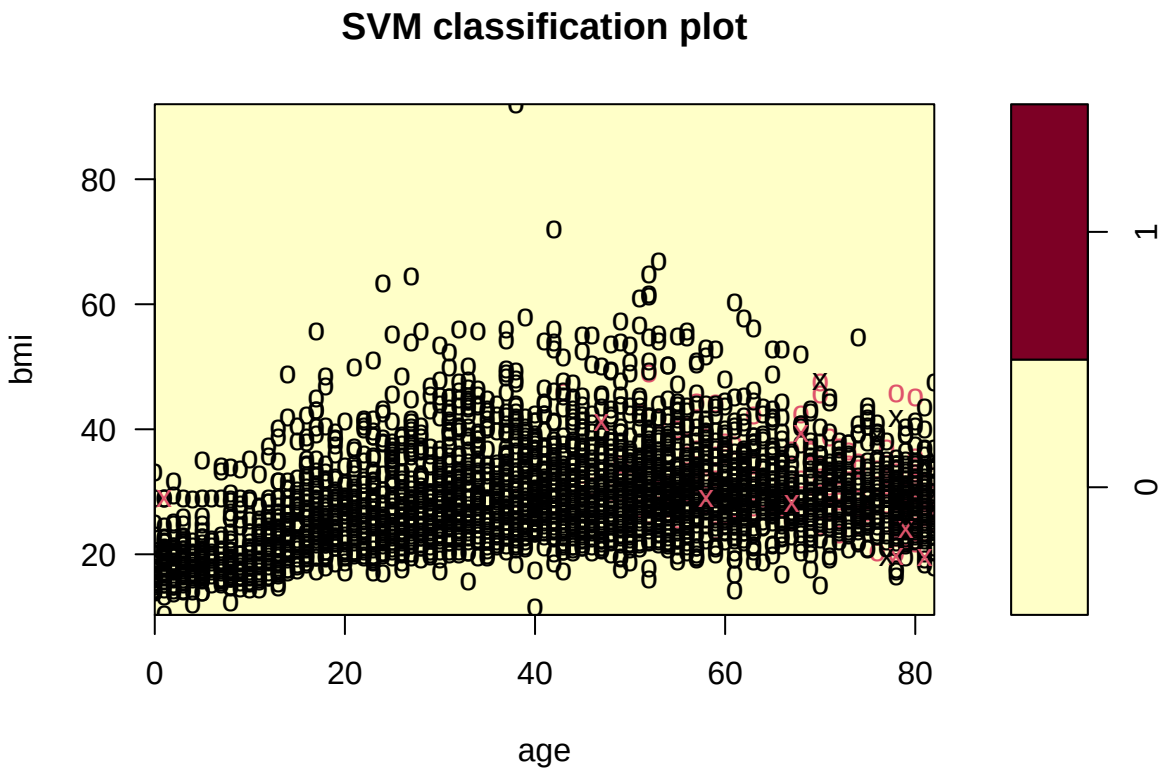
```
##
##    0    1
## 3395  182
```

```r
#svm_model <- svm(stroke ~. , data = df_train_svm, kernel = "linear",scale = TRUE, cost = 10)
#summary(svm_model)
#plot(svm_model, data = df_train_svm, bmi ~ age, slice = list(avg_glucose_level = 3))
```

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

```
##
## Call:
## svm(formula = stroke ~ ., data = df_train_svm, type = "C-classification",
##     kernel = "linear", )
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  21
##
##  ( 8 13 )
##
##
## Number of Classes:  2
##
## Levels:
##  0 1
```

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



In the following we checked the model fit with a confusion matrix with the help of the caret package's confusionMatrix() function.

```
#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
##
```

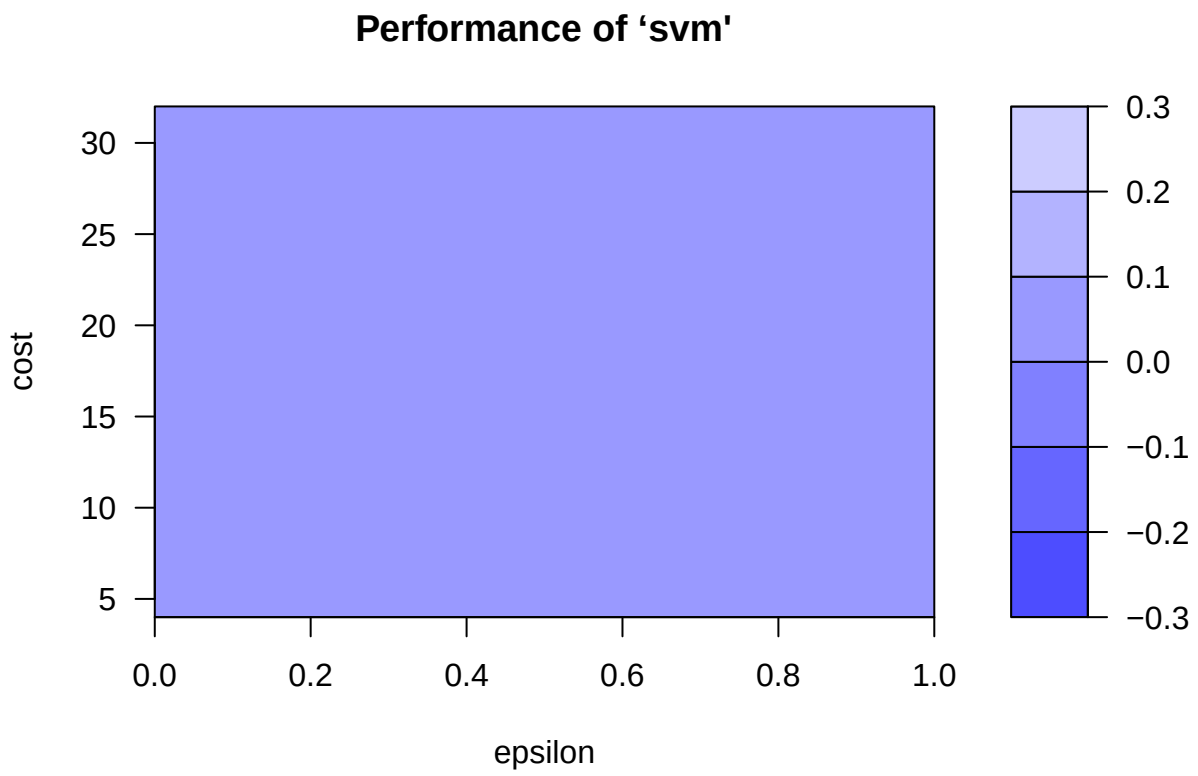
```
svm_test_error
```

```
## [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)
```

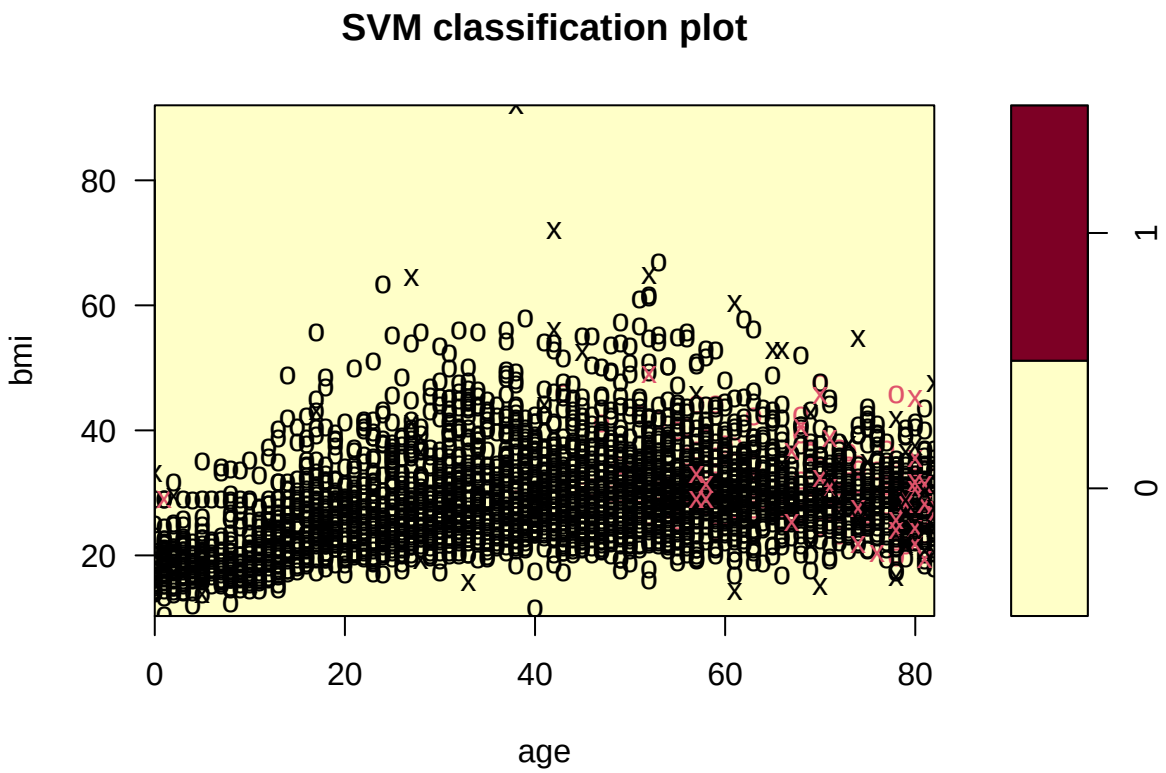
```
##
## Parameter tuning of ’svm’:
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  epsilon cost
##        0    4
##
## - best performance: 0
##
## - Detailed performance results:
##    epsilon cost error dispersion
## 1      0.0    4     0          0
## 2      0.1    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))
```




```
# 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
```

```
#conf_matrix <- confusionMatrix(test_pred, test_truth)
#conf_matrix
```

```
#compare the tuned prediction with the test data
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