# Homework 2 solution

1. Consider the "Auto MPG" which "concerns city-cycle fuel consumption in miles per gallon, to be predicted in terms of 3 multivalued discrete and 5 continuous attributes."[1] The goal is to model mpg given engine displacement and number of cylinders. Answer the following questions.

   a. Load the `autompg.csv` file on Titanium and convert cylinders variable to a factor. (code, output of str())

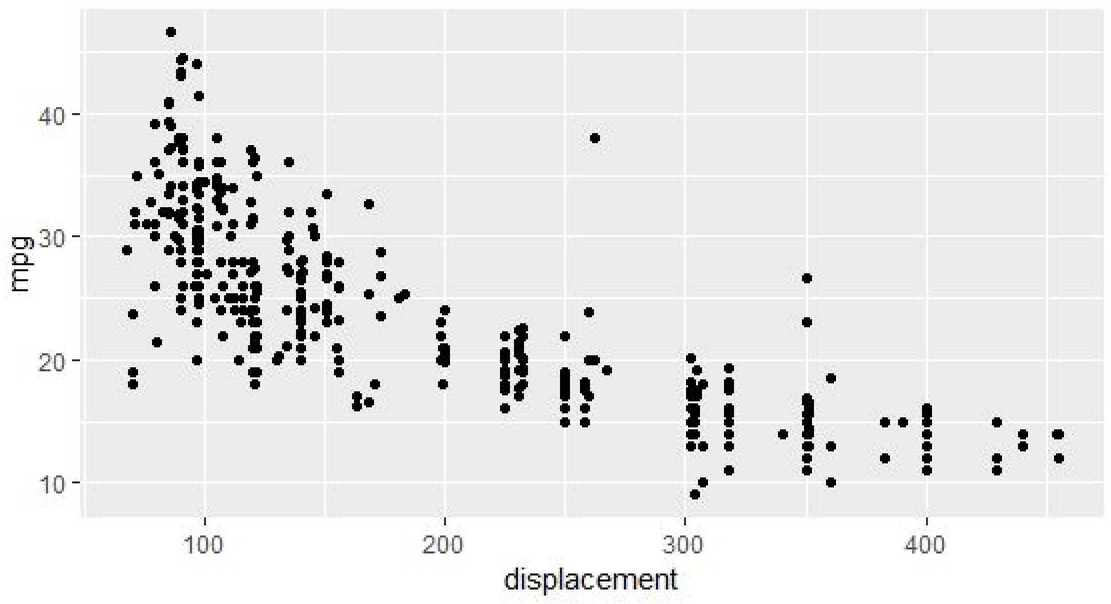
```
> mydata <- read.csv("autompg.csv")
> mydata$cylinders <- as.factor(mydata$cylinders)
> str(mydata)
'data.frame':    398 obs. of  9 variables:
 $ mpg         : num  18 15 18 16 17 15 14 14 14 15 ...
 $ cylinders   : Factor w/ 5 levels "3","4","5","6",..: 5 5 5 5 5 5 5 5 5 5 ...
 $ displacement: num  307 350 318 304 302 429 454 440 455 390 ...
 $ horsepower  : int  130 165 150 150 140 198 220 215 225 190 ...
 $ weight      : int  3504 3693 3436 3433 3449 4341 4354 4312 4425 3850 ...
 $ acceleration: num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
 $ model_year  : int  70 70 70 70 70 70 70 70 70 70 ...
 $ origin      : int  1 1 1 1 1 1 1 1 1 1 …
 $ car_name    : Factor w/ 305 levels "amc ambassador brougham",..: 50 37 232 15 162
142 55 224 242 2 ...
```

   b. Which is the dependent variable? Which are the independent variables?
      Dependent: mpg
      Independent: cylinders, displacement

   c. Plot `mpg` vs. `displacement` (code, plot)

```
ggplot(data=mydata)+geom_point(mapping = aes(x=displacement, y=mpg))
```

[1]Dataset modified from UC Irvine ML repository https://archive.ics.uci.edu/ml/datasets/Auto+MPG. This is *not* the mpg data that is part of the tidyverse datasets.



d. Create a linear model of mpg vs. displacement (only one independent variable). What is the $R^2$? (code, output of summary(model), R2 value)

```
> mod <- lm(data=mydata, mpg~displacement)
> summary(mod)

Call:
lm(formula = mpg ~ displacement, data = mydata)

Residuals:
     Min       1Q   Median       3Q      Max
-12.9550  -3.0569  -0.4928   2.3277  18.6192

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  35.174750   0.491824   71.52   <2e-16 ***
displacement -0.060282   0.002239  -26.93   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.651 on 396 degrees of freedom
Multiple R-squared:  0.6467,  Adjusted R-squared:  0.6459
F-statistic:   725 on 1 and 396 DF,  p-value: < 2.2e-16
```

R2 value = 0.6467

e. Create a new transformed variable that is sqrt(displacement). Create a linear model of mpg vs. sqrt(displacement).

i. Give R code, output of summary(model)

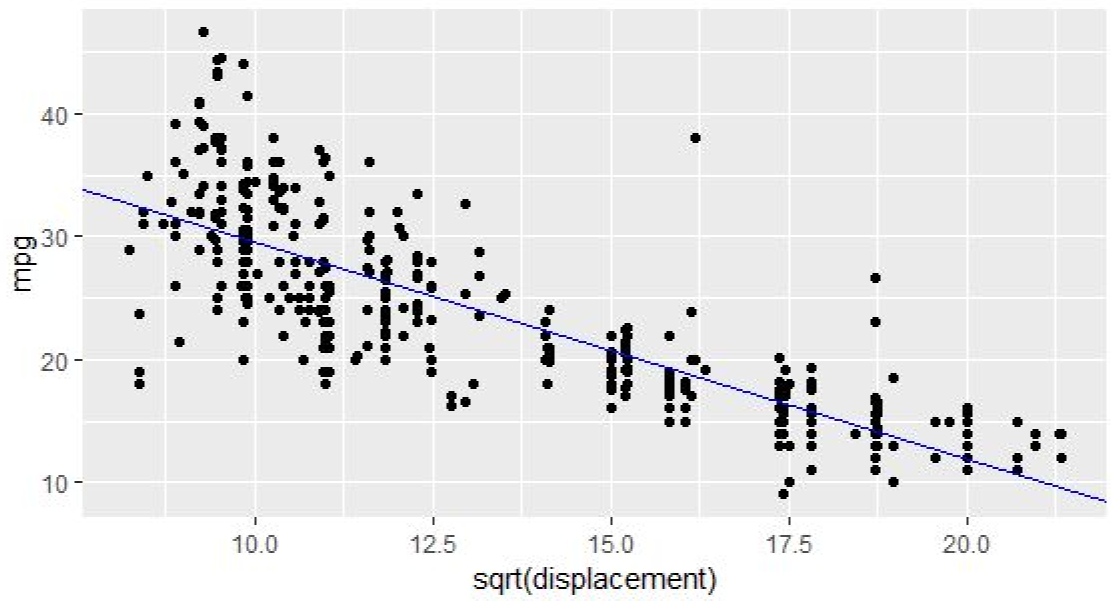
```
> mydata <- mutate(mydata, sqrtdisp=sqrt(displacement))
> mod2 <- lm(data=mydata, mpg~sqrtdisp)
> summary(mod2)

Call:
lm(formula = mpg ~ sqrtdisp, data = mydata)

Residuals:
     Min       1Q   Median       3Q      Max
-14.4479  -2.7327  -0.5354   2.3527  19.3595

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 47.22067    0.85874   54.99   <2e-16 ***
sqrtdisp    -1.76569    0.06175  -28.60   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.47 on 396 degrees of freedom
Multiple R-squared:  0.6737,  Adjusted R-squared:  0.6729
```

ii. Is this a better fit than in part (d)?

Yes, since R2=0.6737 is higher than 0.6467

iii. Plot `mpg` vs. `sqrt(displacement)` and overlay the best fit model as a straight line. (code, plot)

```
coef <- mod2$coefficients
ggplot(data=mydata)+geom_point(mapping = aes(x=sqrtdisp,
y=mpg))+geom_abline(intercept=coef[1], slope=coef[2], color="blue") +
labs(x="sqrt(displacement)")
```

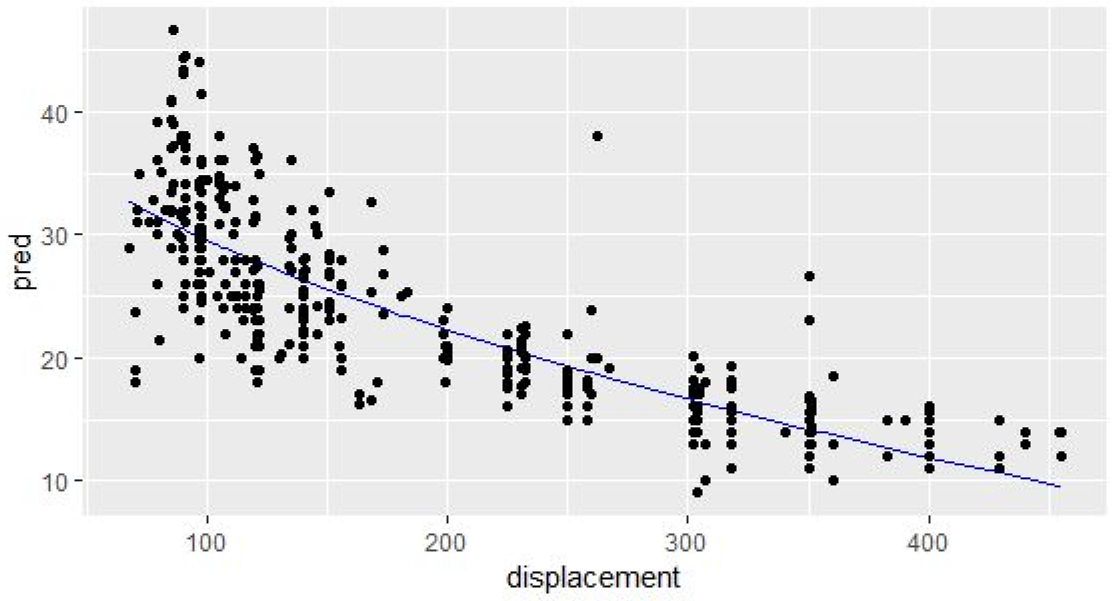


iv. Plot `mpg` vs. `displacement` and overlay the best fit model as a curve. (code, plot) [Hint: plot the predictions; use `add_predictions()` and `geom_line()`. You don't have to use `data_grid()`]

library(modelr)

mydata <- add_predictions(mydata, mod2)

ggplot(data=mydata)+geom_line(mapping = aes(x=displacement, y=pred), color="blue")+geom_point(mapping = aes(x=displacement, y=mpg))



f. Create a linear model of `mpg` vs. `sqrt(displacement)` and `cylinders`.

i. Give R code, output of summary(model)

> mod3 <- lm(data=mydata, mpg~sqrtdisp+cylinders)

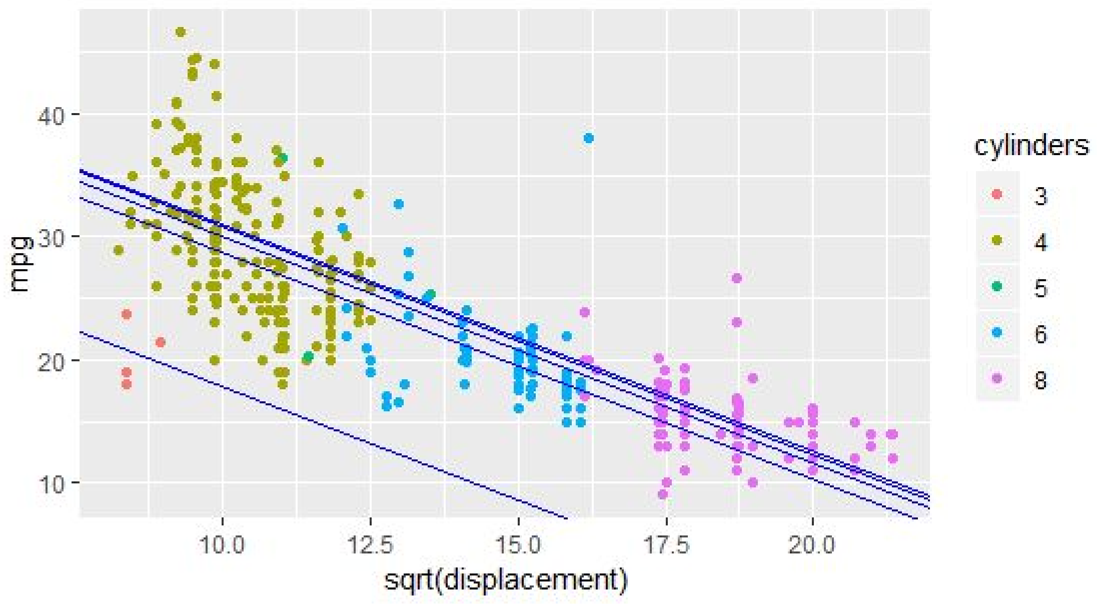
```
> summary(mod3)

Call:
lm(formula = mpg ~ sqrtdisp + cylinders, data = mydata)

Residuals:
     Min       1Q   Median       3Q      Max 
-10.3601  -2.6105  -0.2589   2.1420  20.7138 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)  36.2894     2.7096  13.393  < 2e-16 ***
sqrtdisp     -1.8493     0.1956  -9.452  < 2e-16 ***
cylinders4   12.2841     2.1909   5.607 3.89e-08 ***
cylinders5   13.2524     3.3355   3.973 8.44e-05 ***
cylinders6   10.9303     2.5032   4.367 1.62e-05 ***
cylinders8   12.9472     2.9312   4.417 1.30e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.275 on 392 degrees of freedom
Multiple R-squared:  0.7046,  Adjusted R-squared:  0.7008 
F-statistic:   187 on 5 and 392 DF,  p-value: < 2.2e-16
```

ii. How many dummy (i.e., 0-1) variables were created in the model?
4 dummy variables: cylinders4, cylinders5, cylinders6, cylinders8

iii. Is this a better fit than in part (e)?
Yes, since the adjusted R2 (=0.7008) is higher than the adjusted R2 from (e) (=0.6729)

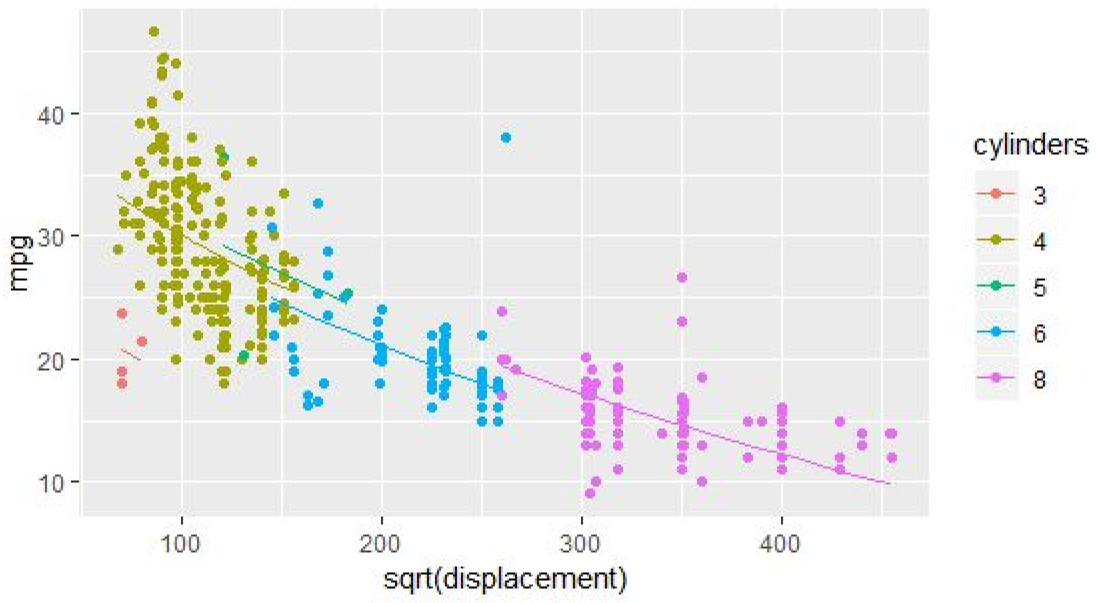
iv. Plot `mpg` vs. `sqrt(displacement)` and overlay the **multiple** linear fit lines: one for each value of the discrete variable. (code, plot)

```
coef <- mod3$coefficients
ggplot(data=mydata2)+geom_point(mapping = aes(x=sqrtdisp, y=mpg,
color=cylinders))+
  geom_abline(intercept=coef[1], slope=coef[2], color="blue") +
  geom_abline(intercept=coef[1]+coef[3], slope=coef[2], color="blue") +
  geom_abline(intercept=coef[1]+coef[4], slope=coef[2], color="blue") +
  geom_abline(intercept=coef[1]+coef[5], slope=coef[2], color="blue") +
  geom_abline(intercept=coef[1]+coef[6], slope=coef[2], color="blue") +
labs(x="sqrt(displacement)")
```



v. Plot `mpg` vs. `displacement` and overlay the best fit model as a curve. (code, plot) [Hint: plot the predictions; use `add_predictions()` and `geom_line()` and use the color aesthetic for cylinders]

```
mydata <- add_predictions(mydata, mod3, var="pred_cyl")
ggplot(data=mydata)+geom_point(mapping = aes(x=displacement, y=mpg, color=cylinders))+
  geom_line(mapping=aes(x=displacement, y=pred_cyl, color=cylinders)) +
labs(x="sqrt(displacement)")
```



2. Consider the toy dataset below which shows if 4 subjects have diabetes or not, along with two diagnostic measurements. [This question is meant to be completed with a calculator; no need to write any R code.]

| **Preg** | **BP** | **HasDiabetes** | **Preg.Norm** | **BP.Norm** |
|---|---|---|---|---|
| 2 | 74 | No | 0.5 | 1 |
| 3 | 58 | Yes | 1 | 0.2 |
| 2 | 58 | Yes | 0.5 | 0.2 |
| 1 | 54 | No | 0 | 0 |
| 2 | 70 | ? | 0.5 | 0.8 |

a. Which variable is the “Class” variable?
   HasDiabetes

b. Normalize the Preg and BP values by scaling the minimum-maximum range of each column to 0-1. Fill in the empty columns in the table.

c. Predict whether a subject with Preg=2, BP=70 will have diabetes using the 1-NN algrotithm and
   i. Using Euclidean distance on the original variables:
      No. (Class of nearest neighbor (2,74); nearest distance=4)
   ii. Using Manhattan distance on the original variables:
      No. (Class of nearest neighbor (2,74); nearest distance=4)
   iii. Using Euclidean distance on the normalized variables:
      No. (Class of nearest neighbor (0.5,1); nearest distance=0.2)
   iv. Using Manhattan distance on the normalized variables:
      No. (Class of nearest neighbor (0.5,1); nearest distance=0.2)

3. The `data_banknote_authentication.csv` file attached on Titanium contains instances of genuine and forged banknotes. The first four columns are features calculated from an industrial camera[2] ; the fifth column indicates if the banknote is forged or not. The goal is to see if it is possible to detect a forgery from only the features.

a. Load and pre-process the data. Show code to:
   i. Load the data file on Titanium.
   ii. How many rows and columns are there?

```
> data_bank <- read.csv("data_banknote_authentication.csv")
> dim(data_bank)
```

[2]For further information, refer to: https://archive.ics.uci.edu/ml/datasets/banknote+authentication

1372 rows, 5 columns

b. Split the dataset into train and test datasets with the *rows 1, 3, 5, ...* for training, and the remaining rows for test (i.e, test using rows 2, 4, 6, …). Do **NOT** randomly sample the data (though resampling is usually done, this hw problem does not use this step for ease of grading). (code)

```
trainedIndex <- seq(1,1372, by=2)
testedIndex <- seq(2,1372, by=2)
trainedLabels <- data_bank[trainedIndex, "forged"]
```

c. Train and test a k-nearest neighbor classifier with the above datasets. *Consider only variance and skewness columns*. Set k=1. What is the error rate (number of misclassifications)? (code)

```
trainedFeatures <- data_bank[trainedIndex, c("variance", "skewness")]
testedFeatures <- data_bank[testedIndex, c("variance", "skewness")]
library(class)
forgedPredict = knn(train = trainedFeatures, cl = trainedLabels, test = testedFeatures,
k=1)
testLabels <- data_bank[testedIndex, "forged"]
table(testLabels, forgedPredict)
          forgedPredict
testLabels   0   1
         0 356  25
         1  15 290
> errorRate <- (25+15)/(356+25+15+290)
> errorRate
[1] 0.05830904
```

d. Repeat part (c) but *consider only variance , skewness, and curtosis columns*. Set k=1. (show code.) What is the error rate? Will the error rate always decrease with larger number of parameters? Why or why not: answer in 2-3 sentences?

```
trainedFeatures <- data_bank[trainedIndex, c("variance", "skewness", "curtosis")]
testedFeatures <- data_bank[testedIndex, c("variance", "skewness", "curtosis")]
forgedPredict = knn(train = trainedFeatures, cl = trainedLabels, test = testedFeatures,
k=1)
table(testLabels, forgedPredict)
  forgedPredict
testLabels   0   1
        0 379   2
        1   0 305

> errorRate <- (2+0)/(686)
> errorRate
```

```
[1] 0.002915452
```

The error does not have to decrease with increasing number of parameters. The additional parameters may be completed irrelevant to the classification task and hence only serve to add noise to the distance calculation.

e. Repeat part (d) but set k=5. What is the error rate?

```
forgedPredict = knn(train = trainedFeatures, cl = trainedLabels, test = testedFeatures, k=5)
table(testLabels, forgedPredict)
           forgedPredict
testLabels   0   1
           0 377   4
           1   2 303
> errorRate <- (2+4)/(686)
> errorRate
[1] 0.008746356
```

f. Repeat part (e) but set k=11. What is the error rate? Considering your observations from (d)-(f), which is the best value for k?

```
forgedPredict = knn(train = trainedFeatures, cl = trainedLabels, test = testedFeatures, k=11)
table(testLabels, forgedPredict)
           forgedPredict
testLabels   0   1
0 377   4
1   1 304

> errorRate <- (1+4)/(356+25+15+290)
> errorRate
[1] 0.00728863
```

From these results, the best value of k is 1 since it gave the lowest error

g. Consider only the ranges of the features - is normalization required?
Yes, normalization is required since the ranges, min and max of the different columns, are different.

h. Normalize each column by scaling the minimum-maximum range of each column to 0-1. (Hint: the built-in R function `scale()` can be used for this) (code)
# or using the normalize() function created in class

```
> normalize <- function(x) {
+   y <- (x - min(x))/(max(x) - min(x))
+ }
> mydataNorm <- mydata %>% mutate(Preg.norm=normalize(Preg))
%>% mutate(Pedigree.norm=normalize(Pedigree)) %>%
mutate(Glucose.norm=normalize(Glucose))
```

i. Train and test a k-nearest neighbor classifier with the normalized dataset. *Consider only variance, skewness, and curtosis columns*. Set k=1. What is the error rate?

```
# using the normalize() function created in class
normalize <- function(x) {
  y <- (x - min(x))/(max(x) - min(x))
}
data_bank <- data_bank %>% mutate(variance.norm=normalize(variance)) %>%
  mutate(skewness.norm=normalize(skewness)) %>%
  mutate(curtosis.norm=normalize(curtosis)) %>%
  mutate(entropy.norm=normalize(entropy))

# or using R's scale function
#Note: apply() executes the given function along every column
maxValues<-apply(data_bank[,1:4], 2, max)
minValues<-apply(data_bank[,1:4], 2, min)
mydataNorm<-scale(data_bank[,1:4], center=minValues, scale=maxValues-minValues)
mydataNorm <- as.data.frame(mydataNorm)
# change column names
names(mydataNorm) <- c("variance.norm", "skewness.norm", "curtosis.norm",
"entropy.norm")
data_bank <- cbind(data_bank, mydataNorm)

# re-create the test and train data sets with normalized columns
trainedFeatures <- data_bank[trainedIndex, c("variance.norm", "skewness.norm",
"curtosis.norm")]
testedFeatures <- data_bank[testedIndex, c("variance.norm", "skewness.norm",
"curtosis.norm")]
forgedPredict = knn(train = trainedFeatures, cl = trainedLabels, test = testedFeatures,
k=1)
table(testLabels, forgedPredict)
        forgedPredict
testLabels   0   1
        0 380   1
        1   0 305
> errorRate <- (1+0)/(380+1+305)
> errorRate
[1] 0.001457726
```