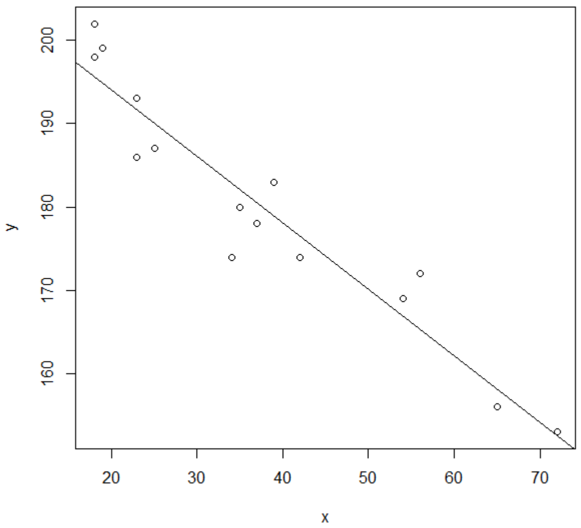
데이터 사이언스 과제3

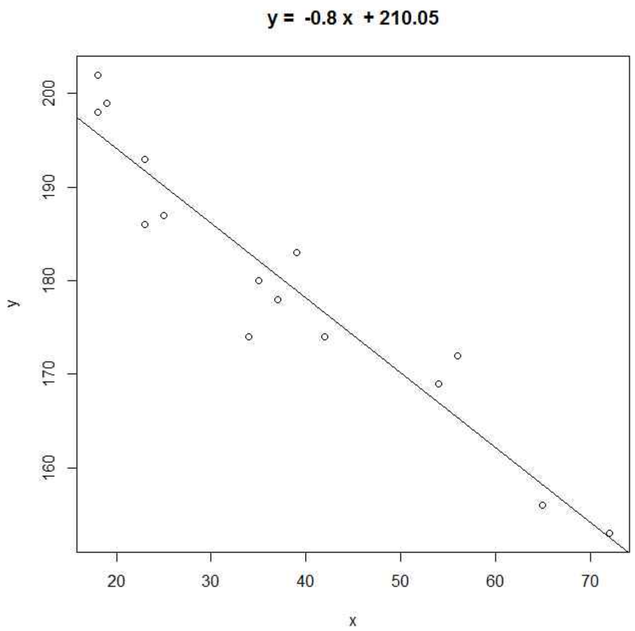
< 1. “과제 02. SimpleR.pdf” 튜토리얼 페이지 100 ~ 121 >

Section 13: Regression Analysis

```
> x = c(18,23,25,35,65,54,34,56,72,19,23,42,18,39,37)
> y = c(202,186,187,180,156,169,174,172,153,199,193,174,198,183,178)
> plot(x,y) # make a plot
> abline(lm(y ~ x)) # plot the regression line
```

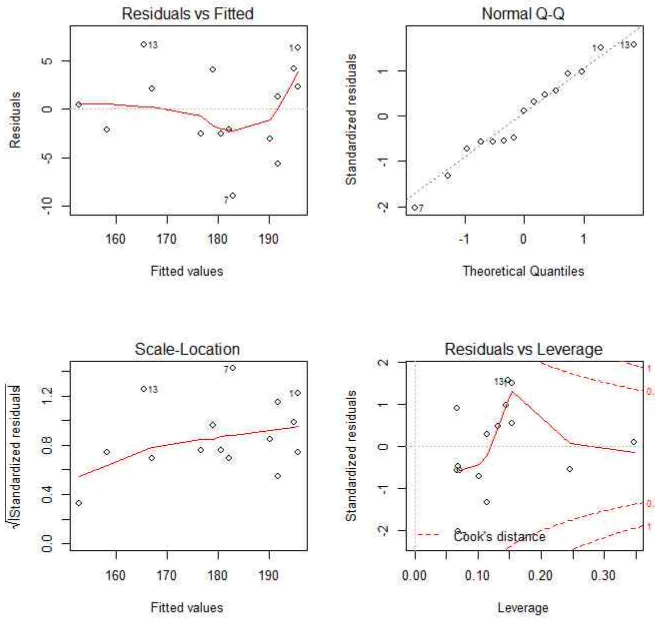


```
> lm(y ~ x)

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
   210.0485      -0.7977
>
> #library(UsingR)
> lm.result=simple.lm(x,y)
```

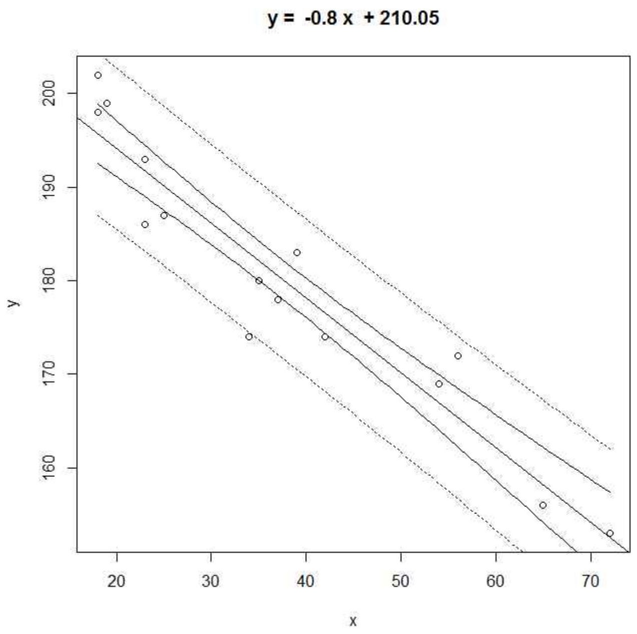


```
> summary(lm.result)

Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-8.9258 -2.5383  0.3879  3.1867  6.6242

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 210.04846    2.86694   73.27  < 2e-16 ***
x            -0.79773    0.06996  -11.40 3.85e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.578 on 13 degrees of freedom
Multiple R-squared:  0.9091,    Adjusted R-squared:  0.9021
F-statistic:   130 on 1 and 13 DF,  p-value: 3.848e-08
>
> coef(lm.result) # or use lm.result[['coef']]
(Intercept)           x
210.0484584  -0.7977266
> lm.res = resid(lm.result) # or lm.result[['resid']]
> summary(lm.res)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-8.9258 -2.5383  0.3879  0.0000  3.1867  6.6242
>
```

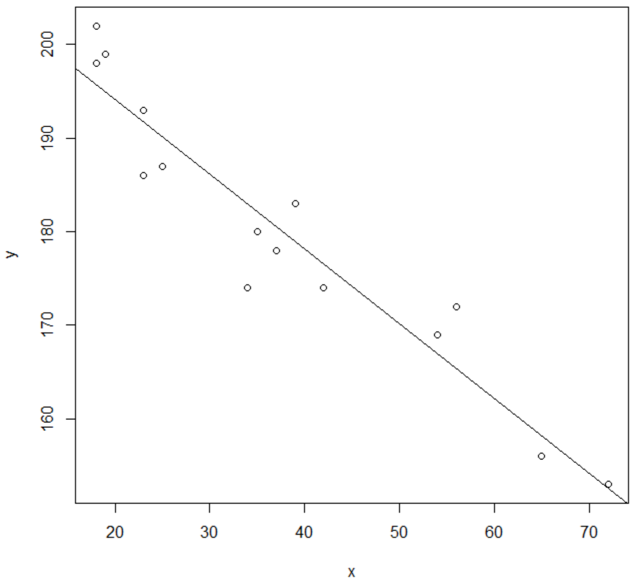
```
> #par(mfrow=c(2,2))
> plot(lm.result)
```



```
> es = resid(lm.result) # the residuals lm.result
> b1 =(coef(lm.result))[['x']] # the x part of the coefficients
> s = sqrt( sum( es^2 ) / (15-2) )
> SE = s/sqrt(sum((x-mean(x))^2))
> t = (b1 - (-1) )/SE # of course - (-1) = +1
> pt(t,13,lower.tail=FALSE) # find the right tail for this value of t
[1] 0.006310157
> # and 15-2 d.f.
>
> SE = s * sqrt( sum(x^2)/( 15*sum((x-mean(x))^2)))
> b0 = 210.04846 # copy or use
> t = (b0 - 220)/SE # (coef(lm.result))[['(Intercept)']]
> pt(t,13,lower.tail=TRUE) # use lower tail (220 or less)
[1] 0.002068424
>
> ## call simple.lm again
> simple.lm(x,y,show.ci=TRUE,conf.level=0.90)

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
   210.0485      -0.7977
```

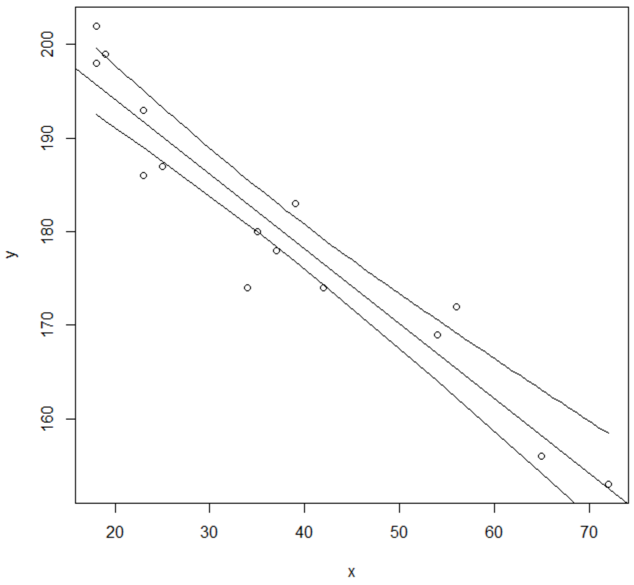


```
> lm.result = lm(y ~ x)
>
> summary(lm.result)

Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-8.9258 -2.5383  0.3879  3.1867  6.6242

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 210.04846    2.86694   73.27  < 2e-16 ***
x            -0.79773    0.06996  -11.40 3.85e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.578 on 13 degrees of freedom
Multiple R-squared:  0.9091,    Adjusted R-squared:  0.9021
F-statistic:   130 on 1 and 13 DF,  p-value: 3.848e-08

>
> plot(x,y)
> abline(lm.result)
```



```
> resid(lm.result)
          1           2           3           4           5           6
7
 6.3106197  -5.7007474  -3.1052943  -2.1280287  -2.1962317   2.0287761
-8.9257552
          8           9          10          11          12          13
14
 6.6242292   0.3878543   4.1083463   1.2992526  -2.5439427   2.3106197
4.0628776
         15
-2.5325755
>
> coef(lm.result)
(Intercept)           x
210.0484584  -0.7977266
>
> coef(lm.result)[1]
(Intercept)
   210.0485
>
> coef(lm.result)['x']
>
> fitted(lm.result) # you can abbreviate to just fitted
       1        2        3        4        5        6        7        8
195.6894 191.7007 190.1053 182.1280 158.1962 166.9712 182.9258 165.3758
       9       10       11       12       13       14       15
152.6121 194.8917 191.7007 176.5439 195.6894 178.9371 180.5326
>
```

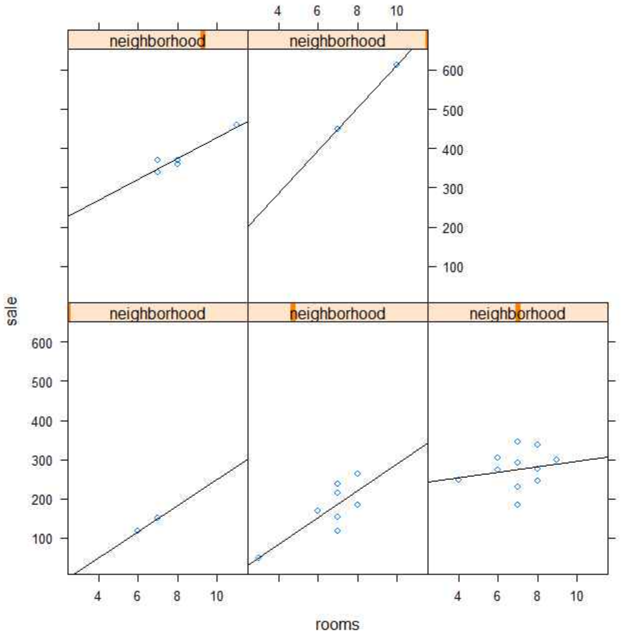
```
> coefficients(lm.result)
(Intercept)           x
210.0484584  -0.7977266
>
> coefficients(summary(lm.result))
               Estimate Std. Error   t value     Pr(>|t|)
(Intercept) 210.0484584 2.86693893  73.26576 2.124074e-18
x            -0.7977266 0.06996281 -11.40215 3.847987e-08
>
> coefficients(summary(lm.result))[2,2]
[1] 0.06996281
>
> coefficients(summary(lm.result))['x','Std. Error']
>
> predict(lm.result,data.frame(x= c(50,60)))
       1        2
170.1621 162.1849
>
> predict(lm.result,data.frame(x=sort(x)), # as before
+   level=.9, interval="confidence") # what is new
        fit      lwr      upr
1  195.6894 192.5083 198.8705
2  195.6894 192.5083 198.8705
3  194.8917 191.8028 197.9805
4  191.7007 188.9557 194.4458
5  191.7007 188.9557 194.4458
6  190.1053 187.5137 192.6969
7  182.9258 180.7922 185.0593
8  182.1280 180.0149 184.2411
9  180.5326 178.4390 182.6262
10 178.9371 176.8337 181.0405
11 176.5439 174.3723 178.7155
12 166.9712 164.0309 169.9116
13 165.3758 162.2564 168.4952
14 158.1962 154.1798 162.2127
15 152.6121 147.8341 157.3902
>
> plot(x,y)
> abline(lm.result)
> ci.lwr = predict(lm.result,data.frame(x=sort(x)),
level=.9,interval="confidence")[,2]
> points(sort(x), ci.lwr,type="l") # or use lines()
>
> curve(predict(lm.result,data.frame(x=x), interval="confidence")[,3],add=T)
```



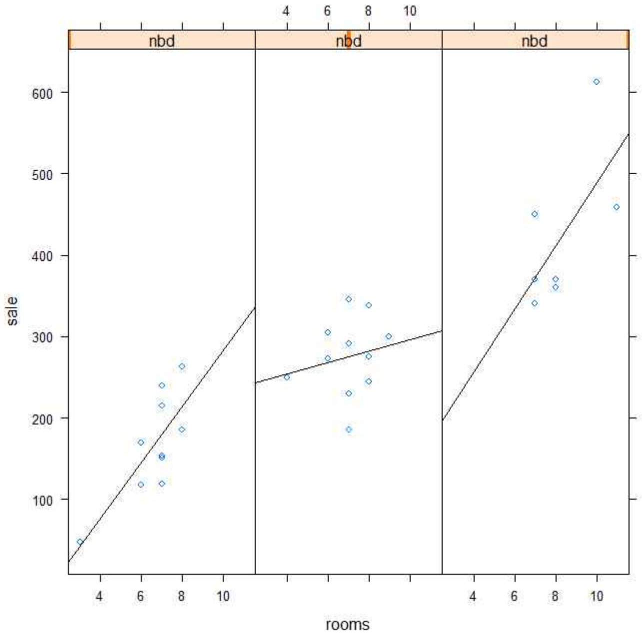
Section 14: Multiple Linear Regression

```
> x = 1:10
> y = sample(1:100,10)
> z = x+y # notice no error term -- sigma = 0
> lm(z ~ x+y) # we use lm() as before

Call:
lm(formula = z ~ x + y)

Coefficients:
(Intercept)            x            y
  4.418e-14    1.000e+00    1.000e+00

> z = x+y + rnorm(10,0,2) # now sigma = 2
> lm(z ~ x+y)

Call:
lm(formula = z ~ x + y)

Coefficients:
(Intercept)            x            y
    -1.8762       0.9644       1.0210

> z = x+y + rnorm(10,0,10) # more noise -- sigma = 10
> lm(z ~ x+y)
```

```
Call:
lm(formula = z ~ x + y)

Coefficients:
(Intercept)            x            y
     9.0834       1.2334       0.7659

>
> lm(z ~ x+y -1) # no intercept beta_0

Call:
lm(formula = z ~ x + y - 1)

Coefficients:
    x      y
1.860  0.859

>
> summary(lm(z ~ x+y ))

Call:
lm(formula = z ~ x + y)

Residuals:
   Min      1Q  Median      3Q     Max
-9.903  -6.639  -3.705   6.764  14.762

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   9.0834    13.3541   0.680  0.51823
x             1.2334     1.2416   0.993  0.35362
y             0.7659     0.1626   4.711  0.00218 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 10.58 on 7 degrees of freedom
Multiple R-squared:  0.764,     Adjusted R-squared:  0.6965
F-statistic: 11.33 on 2 and 7 DF,  p-value: 0.006389

>
> library(lattice); data(homeprice);attach(homeprice)
> panel.lm = function(x,y) {
+   panel.xyplot(x,y)
+   panel.abline(lm(y~x)) }
> xyplot(sale ~ rooms | neighborhood,panel= panel.lm,data=homeprice)
```



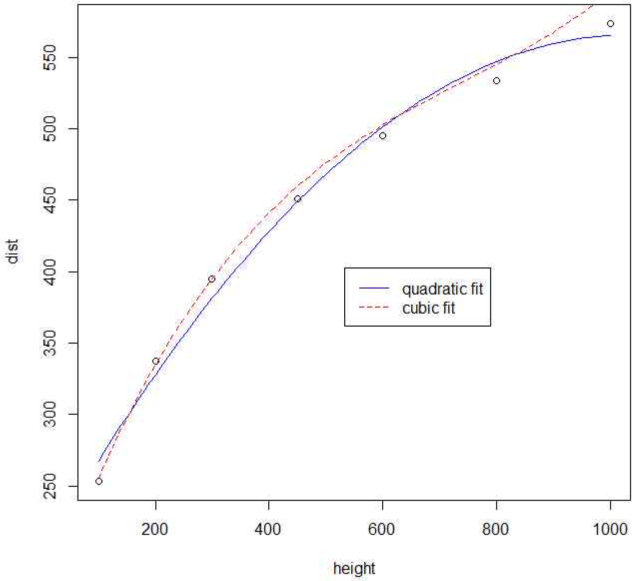
```
> ## too few points in some of the neighborhoods, let's combine
> nbd = as.numeric(cut(neighborhood,c(0,2,3,5),labels=c(1,2,3)))
> table(nbd) # check that we partitioned well
nbd
 1  2  3
10 12  7
> xyplot(sale ~ rooms | nbd, panel= panel.lm,layout=c(3,1))
```



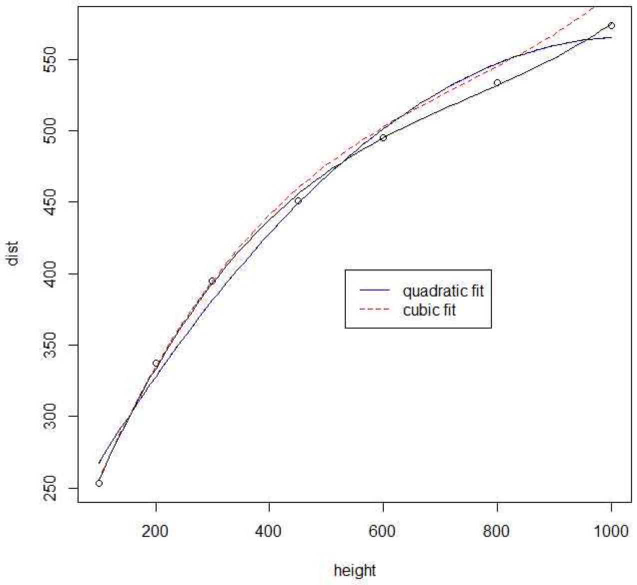
```
> summary(lm(sale ~ bedrooms + nbd))

Call:
lm(formula = sale ~ bedrooms + nbd)

Residuals:
   Min      1Q  Median      3Q     Max
-94.27 -44.27 -14.80   25.73  182.56

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   -58.90      48.54  -1.213   0.2359
bedrooms       35.84      14.94   2.400   0.0239 *
nbd           115.32      15.57   7.405  7.3e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 60.33 on 26 degrees of freedom
Multiple R-squared:  0.7545,     Adjusted R-squared:  0.7356
F-statistic: 39.95 on 2 and 26 DF,  p-value: 1.177e-08

>
> -58.9 + 115.32*(1:3) # nbd is 1, 2 or 3
[1]  56.42 171.74 287.06
>
> summary(lm(sale ~ bedrooms + nbd + full))

Call:
```

```
lm(formula = sale ~ bedrooms + nbd + full)

Residuals:
   Min      1Q  Median      3Q     Max 
-72.85  -43.11  -15.36   22.89  165.38 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)   -67.89      47.58  -1.427   0.1660    
bedrooms       31.74      14.77   2.149   0.0415 *  
nbd           101.00      17.69   5.709 6.04e-06 ***
full           28.51      18.19   1.567   0.1297    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1 

Residual standard error: 58.71 on 25 degrees of freedom
Multiple R-squared:  0.7764,	Adjusted R-squared:  0.7496 
F-statistic: 28.94 on 3 and 25 DF,  p-value: 2.694e-08

> 
> SE = 18.19
> t = (28.51 - 15)/SE
> t
[1] 0.7427158
> pt(t,df=25,lower.tail=F)
[1] 0.232288
> 
> dist = c(253, 337,395,451,495,534,574)
> height = c(100,200,300,450,600,800,1000)
> lm.2 = lm(dist ~ height + I(height^2))
> lm.3 = lm(dist ~ height + I(height^2) + I(height^3))
> lm.2

Call:
lm(formula = dist ~ height + I(height^2))

Coefficients:
(Intercept)       height  I(height^2)  
 200.211950     0.706182    -0.000341  

> lm.3

Call:
lm(formula = dist ~ height + I(height^2) + I(height^3))

Coefficients:
(Intercept)       height  I(height^2)  I(height^3)  
  1.555e+02    1.119e+00   -1.254e-03    5.550e-07  

> 
> #pts = seq(min(height),max(height),length=100)
> quad.fit = 200.211950 + .706182 * pts -0.000341 * pts^2
> cube.fit = 155.5 + 1.119 * pts - .001234 * pts^2 + .000000555 * pts^3
```

```
> plot(height,dist)
> lines(pts,quad.fit,lty=1,col="blue")
> lines(pts,cube.fit,lty=2,col="red")
> legend(locator(1),c("quadratic fit","cubic fit"),lty=1:2,col=c("blue","red"))
```

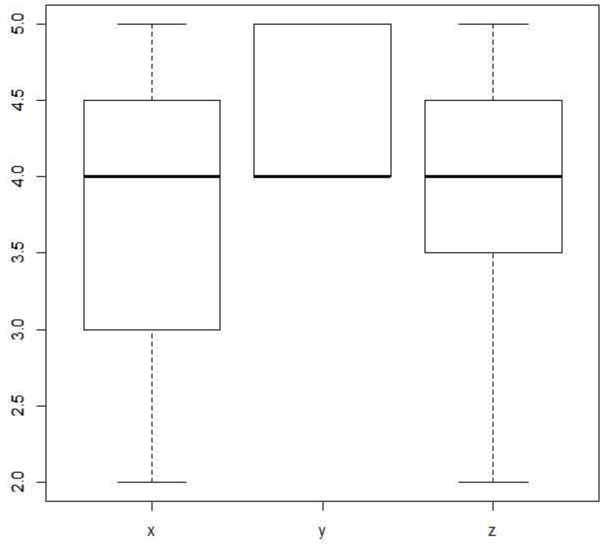


```
> summary(lm.3)

Call:
lm(formula = dist ~ height + I(height^2) + I(height^3))

Residuals:
       1        2        3        4        5        6        7 
-2.35639  3.52782  1.83769 -4.43416  0.01945  2.21560 -0.81001 

Coefficients:
              Estimate Std. Error t value Pr(>|t|)    
(Intercept)  1.555e+02  8.182e+00  19.003 0.000318 ***
height       1.119e+00  6.454e-02  17.332 0.000419 ***
I(height^2) -1.254e-03  1.360e-04  -9.220 0.002699 ** 
I(height^3)  5.550e-07  8.184e-08   6.782 0.006552 ** 
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1 

Residual standard error: 3.941 on 3 degrees of freedom
Multiple R-squared:  0.9994,	Adjusted R-squared:  0.9988 
F-statistic:  1658 on 3 and 3 DF,  p-value: 2.512e-05

> 
> pts = seq(min(height),max(height),length=100)
> makecube = sapply(pts,function(x) coef(lm.3) %*% x^(0:3))
> makesquare = sapply(pts,function(x) coef(lm.2) %*% x^(0:2))
> lines(pts,makecube,lty=1)
> lines(pts,makesquare,lty=2)
```



Section 15: Analysis of Variance

```
> x = c(4,3,4,5,2,3,4,5)
> y = c(4,4,5,5,4,5,4,4)
> z = c(3,4,2,4,5,5,4,4)
> scores = data.frame(x,y,z)
> boxplot(scores)
```



```
> scores = stack(scores) # look at scores if not clear
> names(scores)
[1] "values" "ind"
>
> oneway.test(values ~ ind, data=scores, var.equal=T)

        One-way analysis of means

data:  values and ind
F = 1.1308, num df = 2, denom df = 21, p-value = 0.3417

>
> df = stack(data.frame(x,y,z)) # prepare the data
> oneway.test(values ~ ind, data=df,var.equal=T)

        One-way analysis of means

data:  values and ind
F = 1.1308, num df = 2, denom df = 21, p-value = 0.3417

>
> anova(lm(values ~ ind, data=df))
Analysis of Variance Table

Response: values
          Df Sum Sq Mean Sq F value Pr(>F)
ind        2   1.75 0.87500  1.1308 0.3417
Residuals 21  16.25 0.77381
```

```
>
> kruskal.test(values ~ ind, data=df)

        Kruskal-Wallis rank sum test

data:  values by ind
Kruskal-Wallis chi-squared = 1.9387, df = 2, p-value = 0.3793
```

< 2. Understanding Naive Bayes Classifier Using R >

```
> #Getting started with Naive Bayes
> #Install the package
> install.packages("e1071")
‘C:/Users/stat-513/Documents/R/win-library/3.5’의 위치에 패키지(들)을
설치합니다.
(왜냐하면 ‘lib’가 지정되지 않았기 때문입니다)
URL 'https://cran.csiro.au/bin/windows/contrib/3.5/e1071_1.7-0.zip'을
시도합니다
Content type 'application/zip' length 1015629 bytes (991 KB)
downloaded 991 KB

패키지 ‘e1071’를 성공적으로 압축해제하였고 MD5 sums 이 확인되었습니다

다운로드된 바이너리 패키지들은 다음의 위치에 있습니다

C:\Users\stat-513\AppData\Local\Temp\Rtmp0kV7TH\downloaded_packages
>
> #Loading the library
> library(e1071)
경고메시지(들):
패키지 ‘e1071’는 R 버전 3.5.1에서 작성되었습니다
> ?naiveBayes #Ccontains an example implementation of Titanic dataset
starting httpd help server ... done
> #Next load the Titanic dataset
> data("Titanic")
> str(Titanic)
 'table' num [1:4, 1:2, 1:2, 1:2] 0 0 35 0 0 0 17 0 118 154 ...
 - attr(*, "dimnames")=List of 4
  ..$ Class   : chr [1:4] "1st" "2nd" "3rd" "Crew"
  ..$ Sex     : chr [1:2] "Male" "Female"
  ..$ Age     : chr [1:2] "Child" "Adult"
  ..$ Survived: chr [1:2] "No" "Yes"
> #Save into a data frame and view it
> Titanic_df=as.data.frame(Titanic)
> str(Titanic_df)
'data.frame':   32 obs. of  5 variables:
 $ Class   : Factor w/ 4 levels "1st","2nd","3rd",..: 1 2 3 4 1 2 3 4 1 2 ...
 $ Sex     : Factor w/ 2 levels "Male","Female": 1 1 1 1 2 2 2 2 1 1 ...
 $ Age     : Factor w/ 2 levels "Child","Adult": 1 1 1 1 1 1 1 1 2 2 ...
 $ Survived: Factor w/ 2 levels "No","Yes": 1 1 1 1 1 1 1 1 1 1 ...
 $ Freq    : num  0 0 35 0 0 0 17 0 118 154 ...
```

```
>
> #Creating data from table
> #This will repeat each combination with the frequency of each
combination
> sum(Titanic_df$Freq)
[1] 2201
> repeating_sequence=rep.int(seq_len(nrow(Titanic_df)), Titanic_df$Freq)
> #Create the dataset by row repetition created
> Titanic_dataset=Titanic_df[repeating_sequence,]
> #We no longer need the frequency, drop the feature
> Titanic_dataset$Freq=NULL
>
> #Fitting the Naive Bayes model
> Naive_Bayes_Model=naiveBayes(Survived ~., data=Titanic_dataset)
> #What does the model say? Print the model summary
> Naive_Bayes_Model

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
      No      Yes 
0.676965 0.323035 

Conditional probabilities:
     Class
Y            1st        2nd        3rd       Crew
  No  0.08187919 0.11208054 0.35436242 0.45167785
  Yes 0.28551336 0.16596343 0.25035162 0.29817159

     Sex
Y           Male     Female
  No  0.91543624 0.08456376
  Yes 0.51617440 0.48382560

     Age
Y          Child      Adult
  No  0.03489933 0.96510067
  Yes 0.08016878 0.91983122

>
> #Prediction on the dataset
> NB_Predictions=predict(Naive_Bayes_Model,Titanic_dataset)
> #Confusion matrix to check accuracy
> table(NB_Predictions,Titanic_dataset$Survived)

NB_Predictions   No  Yes
           No  1364  362
           Yes  126  349
>
```

```
> #Getting started with Naive Bayes in mlr
> #Install the package
> install.packages("mlr")
‘C:/Users/stat-513/Documents/R/win-library/3.5’의 위치에 패키지(들)을
설치합니다.
(왜냐하면 ‘lib’가 지정되지 않았기 때문입니다)
‘fastmatch’, ‘ParamHelpers’, ‘BBmisc’, ‘parallelMap’, ‘XML’(들)을 또한
설치합니다.

URL 'https://cran.csiro.au/bin/windows/contrib/3.5/fastmatch_1.1-0.zip'을
시도합니다
Content type 'application/zip' length 53470 bytes (52 KB)
downloaded 52 KB

URL
'https://cran.csiro.au/bin/windows/contrib/3.5/ParamHelpers_1.11.zip'을
시도합니다
Content type 'application/zip' length 433599 bytes (423 KB)
downloaded 423 KB

URL 'https://cran.csiro.au/bin/windows/contrib/3.5/BBmisc_1.11.zip'을
시도합니다
Content type 'application/zip' length 319897 bytes (312 KB)
downloaded 312 KB

URL 'https://cran.csiro.au/bin/windows/contrib/3.5/parallelMap_1.3.zip'을
시도합니다
Content type 'application/zip' length 89960 bytes (87 KB)
downloaded 87 KB

URL 'https://cran.csiro.au/bin/windows/contrib/3.5/XML_3.98-1.16.zip'을
시도합니다
Content type 'application/zip' length 4602444 bytes (4.4 MB)
downloaded 4.4 MB

URL 'https://cran.csiro.au/bin/windows/contrib/3.5/mlr_2.13.zip'을
시도합니다
Content type 'application/zip' length 4689081 bytes (4.5 MB)
downloaded 4.5 MB

패키지 ‘fastmatch’를 성공적으로 압축해제하였고 MD5 sums 이 확인되었습니다
패키지 ‘ParamHelpers’를 성공적으로 압축해제하였고 MD5 sums 이
확인되었습니다
패키지 ‘BBmisc’를 성공적으로 압축해제하였고 MD5 sums 이 확인되었습니다
패키지 ‘parallelMap’를 성공적으로 압축해제하였고 MD5 sums 이 확인되었습니다
패키지 ‘XML’를 성공적으로 압축해제하였고 MD5 sums 이 확인되었습니다
패키지 ‘mlr’를 성공적으로 압축해제하였고 MD5 sums 이 확인되었습니다

다운로드된 바이너리 패키지들은 다음의 위치에 있습니다

C:\Users\stat-513\AppData\Local\Temp\Rtmp0kV7TH\downloaded_packages
> #Loading the library
> library(mlr)
```

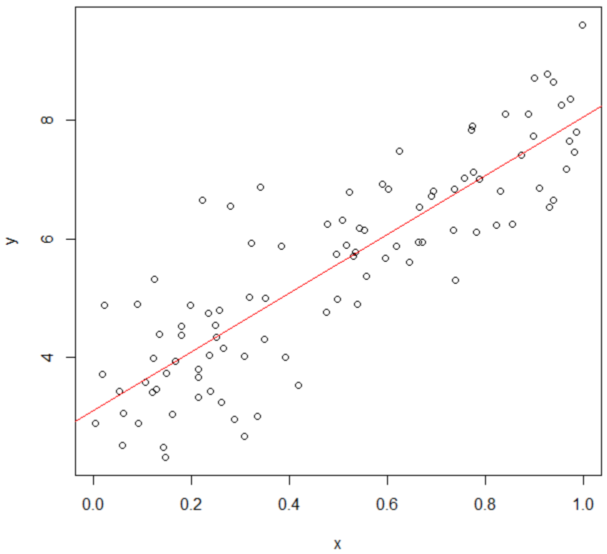
```
필요한 패키지를 로딩중입니다: ParamHelpers

다음의 패키지를 부착합니다: ‘mlr’

The following object is masked from ‘package:e1071’:

    impute

경고메시지(들):
1: 패키지 ‘mlr’는 R 버전 3.5.1에서 작성되었습니다
2: 패키지 ‘ParamHelpers’는 R 버전 3.5.1에서 작성되었습니다
>
> #Create a classification task for learning on Titanic Dataset
> #and specify the target feature
> task = makeClassifTask(data = Titanic_dataset, target = "Survived")
> task
Supervised task: Titanic_dataset
Type: classif
Target: Survived
Observations: 2201
Features:
   numerics     factors     ordered functionals
          0           3           0           0
Missings: FALSE
Has weights: FALSE
Has blocking: FALSE
Has coordinates: FALSE
Classes: 2
  No  Yes
1490  711
Positive class: No
> selected_model = makeLearner("classif.naiveBayes")
> #Train the model
> NB_mlr = train(selected_model, task)
>
> #Read the model learned
> NB_mlr$learner.model

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
      No      Yes
0.676965 0.323035

Conditional probabilities:
     Class
Y            1st        2nd        3rd       Crew
  No  0.08187919 0.11208054 0.35436242 0.45167785
  Yes 0.28551336 0.16596343 0.25035162 0.29817159
```

```
      Sex
Y            Male      Female
  No  0.91543624 0.08456376
  Yes 0.51617440 0.48382560

      Age
Y           Child       Adult
  No  0.03489933 0.96510067
  Yes 0.08016878 0.91983122

>
> #Predict on the dataset without passing the target feature
> predictions_mlr = as.data.frame(predict(NB_mlr, newdata =
Titanic_dataset[,1:3]))
> ##Confusion matrix to check accuracy
> table(predictions_mlr[,1],Titanic_dataset$Survived)

       No  Yes
  No  1364  362
  Yes  126  349
```

< 3. Fitting a Model by Maximum Likelihood >

```
> set.seed(1001)
> N <- 100
> x <- rnorm(N, mean = 3, sd = 2)
> mean(x)
[1] 2.998305
> sd(x)
[1] 2.288979
>
> LL <- function(mu, sigma) {
+    R = dnorm(x, mu, sigma)
+    #
+    -sum(log(R))
+ }
>
> library(stats4)
> mle(LL, start = list(mu = 1, sigma=1))

Call:
mle(minuslogl = LL, start = list(mu = 1, sigma = 1))

Coefficients:
      mu    sigma
2.998305 2.277506
경고메시지(들):
1: In dnorm(x, mu, sigma) : NaN이 생성되었습니다
2: In dnorm(x, mu, sigma) : NaN이 생성되었습니다
3: In dnorm(x, mu, sigma) : NaN이 생성되었습니다
>
> dnorm(x, 1, -1)
  [1] NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
NaN NaN NaN NaN
 [19] NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
NaN NaN NaN NaN
 [37] NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
NaN NaN NaN NaN
 [55] NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
NaN NaN NaN NaN
 [73] NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
NaN NaN NaN NaN
 [91] NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
경고메시지(들):
In dnorm(x, 1, -1) : NaN이 생성되었습니다
>
> mle(LL, start = list(mu = 1, sigma=1), method = "L-BFGS-B", lower =
c(-Inf, 0), upper = c(Inf, Inf))

Call:
mle(minuslogl = LL, start = list(mu = 1, sigma = 1), method = "L-BFGS-B",
    lower = c(-Inf, 0), upper = c(Inf, Inf))

Coefficients:
      mu    sigma
2.998304 2.277506
>
> LL <- function(mu, sigma) {
+    R = suppressWarnings(dnorm(x, mu, sigma))
+    #
+    -sum(log(R))
+ }
> mle(LL, start = list(mu = 1, sigma=1))

Call:
mle(minuslogl = LL, start = list(mu = 1, sigma = 1))

Coefficients:
      mu    sigma
2.998305 2.277506
>
> mle(LL, start = list(mu = 0, sigma=1))

Call:
mle(minuslogl = LL, start = list(mu = 0, sigma = 1))

Coefficients:
      mu    sigma
 51.4840 226.8299
>
> x <- runif(N)
> y <- 5 * x + 3 + rnorm(N)
>
```

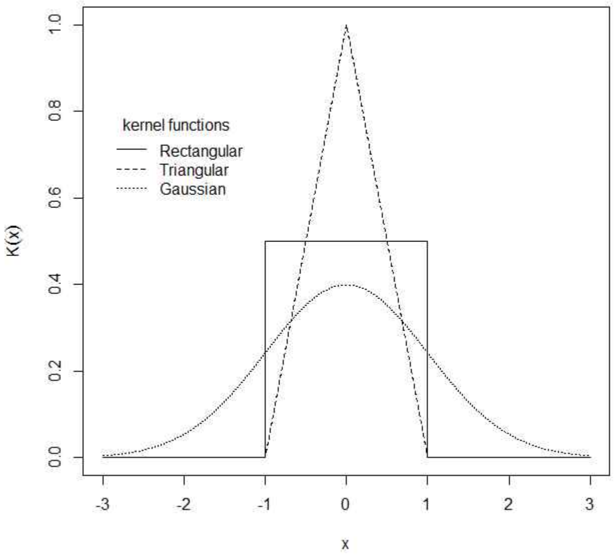
```
> fit <- lm(y ~ x)
> summary(fit)

Call:
lm(formula = y ~ x)

Residuals:
     Min       1Q   Median       3Q      Max 
-1.96206 -0.59016 -0.00166  0.51813  2.43778 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept)   3.1080     0.1695   18.34   <2e-16 ***
x             4.9516     0.2962   16.72   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.8871 on 98 degrees of freedom
Multiple R-squared:  0.7404,	Adjusted R-squared:  0.7378 
F-statistic: 279.5 on 1 and 98 DF,  p-value: < 2.2e-16

> 
> plot(x, y)
> abline(fit, col = "red")
```



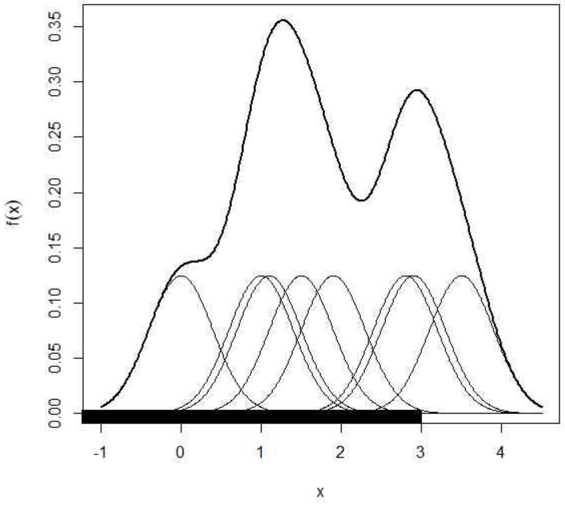
```
> LL <- function(beta0, beta1, mu, sigma) {
+   # Find residuals
+   #
+   R = y - x * beta1 - beta0
+   #
+   # Calculate the likelihood for the residuals (with mu and sigma as
parameters)
+   #
+   R = suppressWarnings(dnorm(R, mu, sigma))
+   #
+   # Sum the log likelihoods for all of the data points
+   #
+   -sum(log(R))
+ }
> 
> LL <- function(beta0, beta1, mu, sigma) {
+   R = y - x * beta1 - beta0
+   #
+   R = suppressWarnings(dnorm(R, mu, sigma, log = TRUE))
+   #
+   -sum(R)
+ }
> 
> fit <- mle(LL, start = list(beta0 = 3, beta1 = 1, mu = 0, sigma=1))
Error in solve.default(oout$hessian) : 
  Lapack routine dgesv: system is exactly singular: U[4,4] = 0
> 
> fit <- mle(LL, start = list(beta0 = 5, beta1 = 3, mu = 0, sigma=1))
> # OK
> fit

Call:
mle(minuslogl = LL, start = list(beta0 = 5, beta1 = 3, mu = 0, 
    sigma = 1))

Coefficients:
     beta0      beta1         mu      sigma 
 4.0540205  4.9516167 -0.9459795  0.8782279 
> 
> fit <- mle(LL, start = list(beta0 = 4, beta1 = 2, mu = 0, sigma=1))
Error in solve.default(oout$hessian) : 
  system is computationally singular: reciprocal condition number = 
1.68198e-21
> fit

Call:
mle(minuslogl = LL, start = list(beta0 = 5, beta1 = 3, mu = 0, 
    sigma = 1))

Coefficients:
     beta0      beta1         mu      sigma 
 4.0540205  4.9516167 -0.9459795  0.8782279 
> 
> summary(fit)
Maximum likelihood estimation

Call:
mle(minuslogl = LL, start = list(beta0 = 5, beta1 = 3, mu = 0, 
    sigma = 1))

Coefficients:
        Estimate Std. Error
beta0   4.0540205        NaN
beta1   4.9516167 0.29319257
mu     -0.9459795        NaN
sigma   0.8782279 0.06209982

-2 log L: 257.8177 
경고메시지(들): 
In sqrt(diag(object@vcov)) : NaN이 생성되었습니다
> 
> fit <- mle(LL, start = list(beta0 = 2, beta1 = 1.5, sigma=1), fixed = list(mu 
= 0), nobs = length(y))
> summary(fit)
Maximum likelihood estimation

Call:
mle(minuslogl = LL, start = list(beta0 = 2, beta1 = 1.5, sigma = 1), 
    fixed = list(mu = 0), nobs = length(y))

Coefficients:
       Estimate Std. Error
beta0 3.1080361 0.16779400
beta1 4.9516269 0.29319183
sigma 0.8782257 0.06209942

-2 log L: 257.8177 
> 
> AIC(fit)
[1] 263.8177
> BIC(fit)
[1] 271.6332
> logLik(fit)
'log Lik.' -128.9088 (df=3)
> 
> #install.packages("bbmle")
> library(bbmle)
경고메시지(들): 
패키지 'bbmle'는 R 버전 3.5.1에서 작성되었습니다 
> fit <- mle2(LL, start = list(beta0 = 3, beta1 = 1, mu = 0, sigma = 1))
> summary(fit)
Maximum likelihood estimation

Call:
mle2(minuslogl = LL, start = list(beta0 = 3, beta1 = 1, mu = 0, 
    sigma = 1))

Coefficients:
      Estimate Std. Error z value  Pr(z)    
beta0 3.054021   0.083897 36.4019 <2e-16 ***
beta1 4.951617   0.293193 16.8886 <2e-16 ***
mu    0.054021   0.083897  0.6439 0.5196    
sigma 0.878228   0.062100 14.1421 <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

-2 log L: 257.8177 
```

< 4. Density Estimation >

```
> #7.1 Introduction
> #7.2 Density Estimation
> x <- c(0, 1, 1.1, 1.5, 1.9, 2.8, 2.9, 3.5)
> n <- length(x)
> xgrid <- seq(from = min(x) - 1, to = max(x) + 1, by = 0.01)
> h <- 0.4
> bumps <- sapply(x, function(a) gauss((xgrid - a)/h)/(n * h))
> 
> #7.3 Analysis Using R
> logL <- function(param, x) {
+   d1 <- dnorm(x, mean = param[2], sd = param[3])
+   d2 <- dnorm(x, mean = param[4], sd = param[5])
+   -sum(log(param[1] * d1 + (1 - param[1]) * d2))
+ }
> startparam <- c(p = 0.5, mu1 = 50, sd1 = 3, mu2 = 80, sd2 = 3)
> opp <- optim(startparam, logL, x = faithful$waiting)
11건의 경고들이 발견되었습니다 (이를 확인하기 위해서는 warnings()를
이용하시길 바랍니다).
> rec <- function(x) (abs(x) < 1) * 0.5
> tri <- function(x) (abs(x) < 1) * (1 - abs(x))
> gauss <- function(x) 1/sqrt(2*pi) * exp(-(x^2)/2)
> x <- seq(from = -3, to = 3, by = 0.001)
> plot(x, rec(x), type = "l", ylim = c(0,1), lty = 1, ylab = expression(K(x)))
> lines(x, tri(x), lty = 2)
> lines(x, gauss(x), lty = 3)
> legend(-3, 0.8, legend = c("Rectangular", "Triangular", "Gaussian"), lty = 
1:3,
+   title = "kernel functions", bty = "n")
```

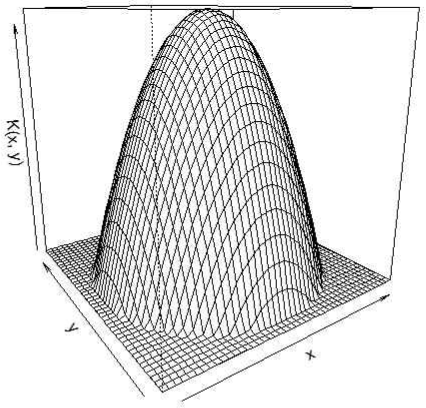


```
> plot(xgrid, rowSums(bumps), ylab = expression(hat(f)(x)), type = "l", xlab =
"x", lwd = 2)
> rug(x, lwd = 2)
경고메시지(들):
In rug(x, lwd = 2) : 일부 값들이 잘려나갈 것입니다
> out <- apply(bumps, 2, function(b) lines(xgrid, b))
```

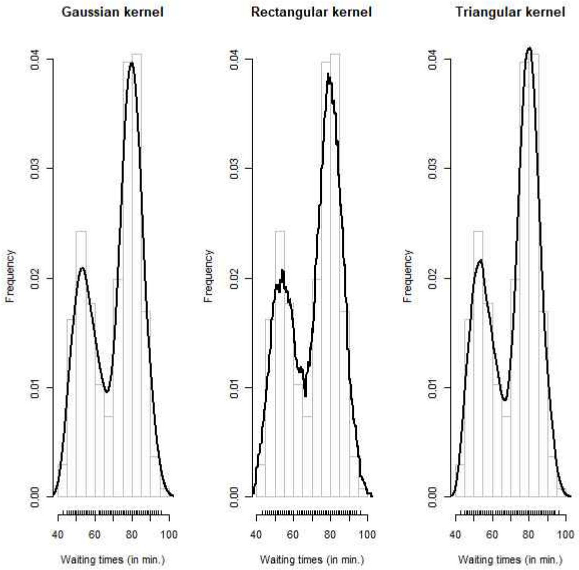


```
>    #method = "L-BFGS-B",
```

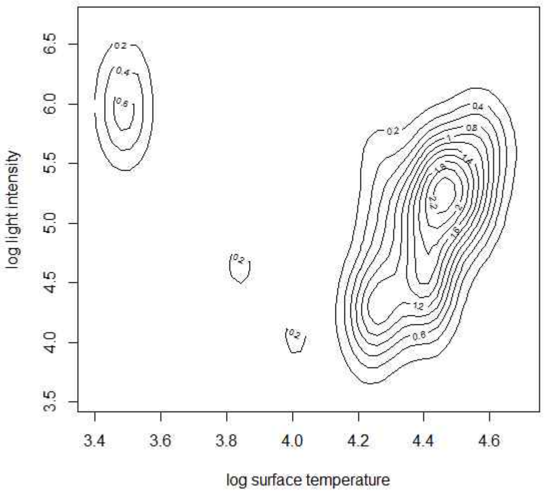
```
>    #lower = c(0.01, rep(1, 4)),
>    #upper = c(0.99, rep(200, 4)))
> opp
$par
        p       mu1       sd1       mu2       sd2 
0.3608441 54.6141258  5.8703887 80.0908981  5.8683161 

$value
[1] 1034.002

$counts
function gradient 
     323       NA 

$convergence
[1] 0

$message
NULL

>
> epa <- function(x, y)
+    ((x^2 + y^2) < 1) * 2/pi * (1 - x^2 - y^2)
> x <- seq(from = -1.1, to = 1.1, by = 0.05)
> epavals <- sapply(x, function(a) epa(a, x))
> persp(x = x, y = x, z = epavals, xlab = "x", ylab = "y",
+    zlab = expression(K(x, y)), theta = -35, axes = TRUE,
+    box = TRUE)
```



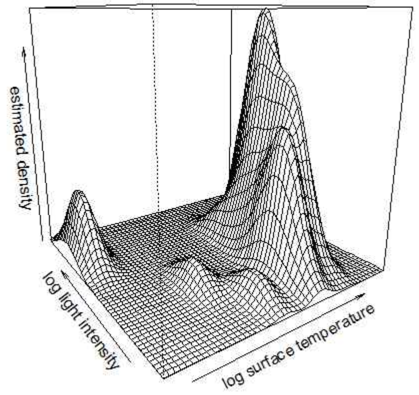
```
> data("faithful", package = "datasets")
> x <- faithful$waiting
> layout(matrix(1:3, ncol = 3))
> hist(x, xlab = "Waiting times (in min.)", ylab = "Frequency",
+    probability = TRUE, main = "Gaussian kernel",
+    border = "gray")
> lines(density(x, width = 12), lwd = 2)
> rug(x)
> hist(x, xlab = "Waiting times (in min.)", ylab = "Frequency",
+    probability = TRUE, main = "Rectangular kernel",
+    border = "gray")
> lines(density(x, width = 12, window = "rectangular"), lwd = 2)
> rug(x)
> hist(x, xlab = "Waiting times (in min.)", ylab = "Frequency",
+    probability = TRUE, main = "Triangular kernel",
+    border = "gray")
> lines(density(x, width = 12, window = "triangular"), lwd = 2)
> rug(x)
```



```
> #install.packages("KernSmooth")
> #install.packages("HSAUR")
> library("KernSmooth")
> data("CYGOB1", package = "HSAUR")
> CYGOB1d <- bkde2D(CYGOB1, bandwidth = sapply(CYGOB1, dpik))
> contour(x = CYGOB1d$x1, y = CYGOB1d$x2, z = CYGOB1d$fhat,
+    xlab = "log surface temperature",
+    ylab = "log light intensity")
```



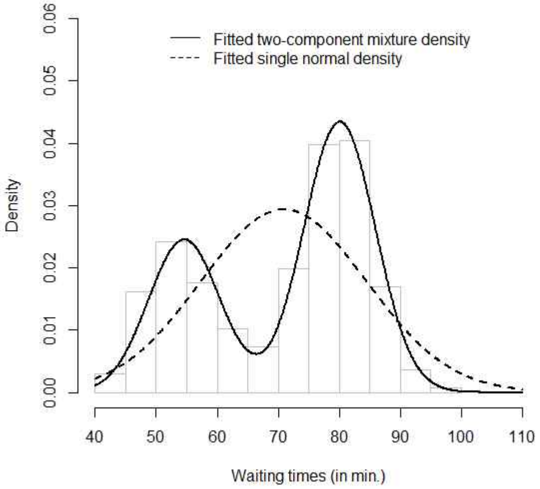
```
> persp(x = CYGOB1d$x1, y = CYGOB1d$x2, z = CYGOB1d$fhat,
+    xlab = "log surface temperature",
+    ylab = "log light intensity",
+    zlab = "estimated density",
+    theta = -35, axes = TRUE, box = TRUE)
```



```
> #install.packages("mclust")
> library("mclust")
> mc <- Mclust(faithful$waiting)
fitting ...
```

```
  |
  |                                                                      |
0%
  |
  |====                                                                  |
5%
  |
  |=======                                                               |
11%
  |
  |===========                                                           |

(중략)

|  89%
  |
  |==================================================================
|  95%
  |

|======================================================================|
100%
> mc
'Mclust' model object: (E,2)

Available components:
 [1] "call"           "data"           "modelName"      "n"
 [5] "d"              "G"              "BIC"            "bic"
 [9] "loglik"         "df"             "hypvol"         "parameters"
[13] "z"              "classification" "uncertainty"
>
> mc$parameters$mean
       1        2
54.61675 80.09239
>
> sqrt(mc$parameters$variance$sigmasq)
[1] 5.868639
>
> #install.packages("flexmix")
> library("flexmix")
필요한 패키지를 로딩중입니다: lattice
> fl <- flexmix(waiting ~ 1, data = faithful, k = 2)
>
> parameters(fl, component = 1)
                    Comp.1
coef.(Intercept) 71.04531
sigma            13.55347
>
> parameters(fl, component = 2)
                    Comp.2
coef.(Intercept) 70.70241
sigma            13.64683
>
```

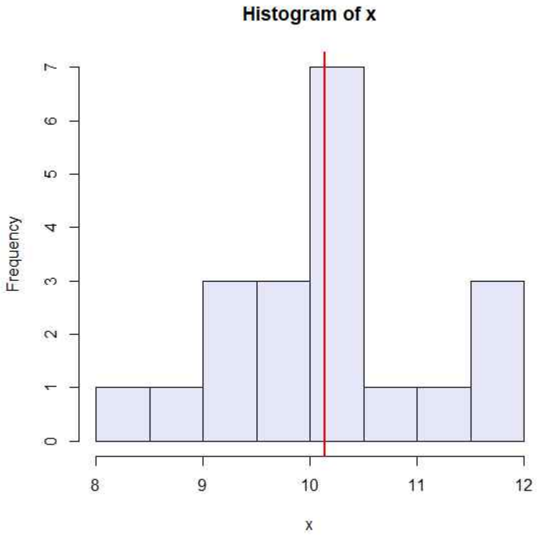
```
> library("boot")
> fit <- function(x, indx) {
+   a <- Mclust(x[indx], minG = 2, maxG = 2)$parameters
+   if (a$pro[1] < 0.5) return(c(p = a$pro[1], mu1 = a$mean[1]))
+ }
>
> opar <- as.list(opp$par)
> rx <- seq(from = 40, to = 110, by = 0.1)
> d1 <- dnorm(rx, mean = opar$mu1, sd = opar$sd1)
> d2 <- dnorm(rx, mean = opar$mu2, sd = opar$sd2)
> f <- opar$p * d1 + (1 - opar$p) * d2
> hist(x, probability = TRUE, xlab = "Waiting times (in min.)",
+   border = "gray", xlim = range(rx), ylim = c(0, 0.06),
+   main = "")
> lines(rx, f, lwd = 2)
> lines(rx, dnorm(rx, mean = mean(x), sd = sd(x)), lty = 2, lwd = 2)
> legend(50, 0.06, lty = 1:2, bty = "n",
+   legend = c("Fitted two-component mixture density", "Fitted single
normal density"))
```



```
>   #mu2 = a$mean[2]))
>   #return(c(p = 1 - a$pro[1], mu1 = a$mean[2],
>   #mu2 = a$mean[1]))
> #}
>
> bootpara <- boot(faithful$waiting, fit, R = 1000)
fitting ...

  |
  |                                                                      |
0%
  |
  |====                                                                  |
5%
  |
  |=======                                                          |
11%
  |

(중략)

84%
  |
  |================================================================
|  89%
  |
  |===================================================================
|  95%
  |

|==============================================================================|
100%
>
> boot.ci(bootpara, type = "bca", index = 1)
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = bootpara, type = "bca", index = 1)

Intervals :
Level       BCa
95%   ( 0.2976,  0.4256 )
Calculations and Intervals on Original Scale
>
> boot.ci(bootpara, type = "bca", index = 2)
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = bootpara, type = "bca", index = 2)

Intervals :
Level       BCa
95%   (53.02, 55.92 )
Calculations and Intervals on Original Scale
>
> boot.ci(bootpara, type = "bca", index = 3)
Error in boot.out$t[, index] : 첨자의 허용 범위를 벗어났습니다
>
> bootplot <- function(b, index, main = "") {
+   dens <- density(b$t[,index])
+   ci <- boot.ci(b, type = "bca", index = index)$bca[4:5]
+
+ layout(matrix(1:2, ncol = 2))
+ bootplot(bootpara, 2, main = expression(mu[1]))
+ bootplot(bootpara, 3, main = expression(mu[2]))
+
+   est <- b$t0[index]
+   plot(dens, main = main)
+   y <- max(dens$y) / 10
+   segments(ci[1], y, ci[2], y, lty = 2)
+   points(ci[1], y, pch = "(")
+   points(ci[2], y, pch = ")")
+   points(est, y, pch = 19)
+ }
```

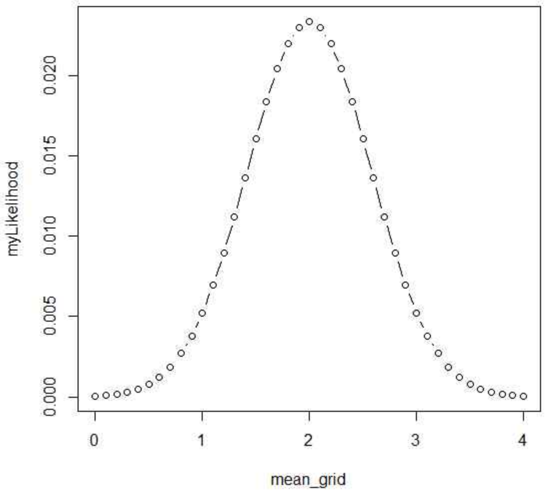
< 5. Understanding the EM (Expectation Maximization) Algorithm >

```
> set.seed(123) ## ensures we all see the same output
> trueMean <- 10 ## suppose this true mean is unknown
> n <- 20
> x <- rnorm(n, mean = trueMean) ## sample data from a Normal
distribution
> print(x)
 [1]  9.439524  9.769823 11.558708 10.070508 10.129288 11.715065
10.460916
 [8]  8.734939  9.313147  9.554338 11.224082 10.359814 10.400771
10.110683
[15]  9.444159 11.786913 10.497850  8.033383 10.701356  9.527209
> hist(x, col = "lavender")
> abline(v = mean(x), col = "red", lwd = 2) ## highlight sample mean
```

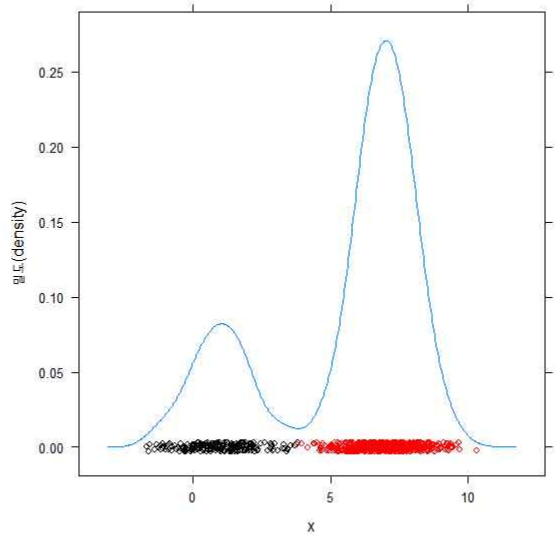


```
> dat <- c(1,2,3) # mean of 'dat' is 2
> rbind(prod(dnorm(dat, mean=1.5, sd=1)),
+   prod(dnorm(dat, mean=2, sd=1)),
+   prod(dnorm(dat, mean=2.5, sd=1)))
```

```
             [,1]
[1,] 0.01605371
[2,] 0.02335800
[3,] 0.01605371
>
> dat <- c(1,2,3)
> mean_grid <- seq(0, 4, by=0.1) ## values of the mean to check the
likelihood at
> myLikelihood <- rep(0, length(mean_grid) )
> for( i in seq_along( myLikelihood ) ) {
+   myLikelihood[i] <- prod( dnorm( dat, mean = mean_grid[i], sd=1 ) )
+   }
> plot( myLikelihood ~ mean_grid, type="b" )
```



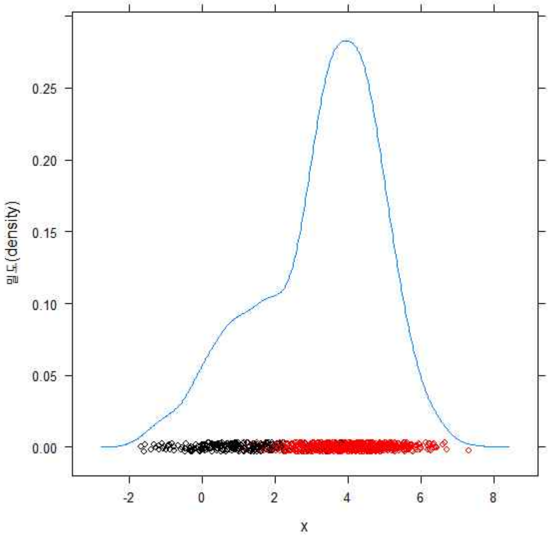
```
> set.seed(123)
> tau_1_true <- 0.25
> x <- y <- rep(0,1000)
> for( i in 1:1000 ) {
+   if( runif(1) < tau_1_true ) {
+     x[i] <- rnorm(1, mean=1) # 25%의 확률로 head가 나오면 정규분포
D1에서 샘플링
+     y[i] <- "heads"
+   } else {
+     x[i] <- rnorm(1, mean=7) # 75%의 확률로 tail이 나오면 정규분포 D2에서
샘플링
+     y[i] <- "tails"
+   }
+ }
> library(“lattice”)
> densityplot( ~x, par.settings = list( plot.symbol = list(col=as.factor(y))))
```



```
>
> print( x[1] )## 랜덤변수 값으로 변동
[1] 7.800554
> dnorm( x[1], mean=0 )## x[1]의 값에 따라 변동
[1] 2.442273e-14
> dnorm( x[1], mean=1 ) ## x[1]의 값에 따라 변동
[1] 3.617301e-11
>
> tau_1 <- 0.5 ## our initial believed proportion from D1, chosen
arbitrarily
> tau_2 <- 0.5 ## our initial believed proportion from D2, chosen
arbitrarily
> T_1 <- tau_1 * dnorm( x[1], mean=0 )
> T_2 <- tau_2 * dnorm( x[1], mean=1 )
> print( T_1 ) ## x[1]의 값에 따라 변동
[1] 1.221137e-14
> print( T_2 ) ## x[1]의 값에 따라 변동
[1] 1.808651e-11
> T_1 / (T_1 + T_2) ## x[1]의 값에 따라 변동
[1] 0.0006747089
>
> T_1 <- tau_1 * dnorm( x, mean=0 )
> T_2 <- tau_2 * dnorm( x, mean=1 )
> head( T_1 / (T_1 + T_2) )
[1] 0.0006747089 0.0003162351 0.0004351016 0.0002704738 0.0012503475
[6] 0.0029791099
>
> P_1 <- T_1 / (T_1 + T_2)
> P_2 <- T_2 / (T_1 + T_2)
> mu_1 <- sum( P_1 * x ) / sum(P_1)
> mu_2 <- sum( P_2 * x ) / sum(P_2)
> c(mu_1, mu_2)
[1] 0.5045618 6.1011529
```

```
>
> ## set the initial guesses for the distribution parameters
> mu_1 <- 0
> mu_2 <- 1
> ## as well as the latent variable parameters
> tau_1 <- 0.5
> tau_2 <- 0.5
> for( i in 1:10 ) {
+   ## Given the observed data, as well as the distribution parameters,
+   ## what are the latent variables?
+   T_1 <- tau_1 * dnorm( x, mu_1 )
+   T_2 <- tau_2 * dnorm( x, mu_2 )
+   P_1 <- T_1 / (T_1 + T_2)
+   P_2 <- T_2 / (T_1 + T_2) ## note: P_2 = 1 - P_1
+   tau_1 <- mean(P_1)
+   tau_2 <- mean(P_2)
+   ## Given the observed data, as well as the latent variables,
+   ## what are the population parameters?
+   mu_1 <- sum( P_1 * x ) / sum(P_1)
+   mu_2 <- sum( P_2 * x ) / sum(P_2)
+   ## print the current estimates
+   print( c(mu_1, mu_2, mean(P_1)) )
+ }
[1] 0.5045618 6.1011529 0.1002794
[1] 0.8546336 6.9403680 0.2301181
[1] 0.9732251 7.0006108 0.2423406
[1] 0.9853947 7.0054109 0.2434347
[1] 0.9864849 7.0058260 0.2435309
[1] 0.9865811 7.0058624 0.2435394
[1] 0.9865895 7.0058656 0.2435401
[1] 0.9865903 7.0058659 0.2435402
[1] 0.9865903 7.0058660 0.2435402
[1] 0.9865904 7.0058660 0.2435402
>
> #install.packages("mixtools") ## if you don't have it already.
> library("mixtools")
mixtools package, version 1.1.0, Released 2017-03-10
This package is based upon work supported by the National Science
Foundation under Grant No. SES-0518772.

> myEM <- normalmixEM( x, mu = c(0,1), sigma=c(1,1), sd.constr=c(1,1) )
number of iterations= 7
> ## number of iterations= 7
> myEM$mu ## the distribution means
[1] 0.9865898 7.0058658
> ## [1] 0.9866 7.0059
> myEM$lambda ## the mixing probabilities
[1] 0.2435402 0.7564598
> ## [1] 0.2435 0.7565
>
> set.seed(123)
> tau_true <- 0.25
> x <- y <- rep(0,1000)
```

```
> for( i in 1:1000 ) {
+   if( runif(1) < tau_true ) {
+     x[i] <- rnorm(1, mean=1)
+     y[i] <- "heads"
+   } else {
+     x[i] <- rnorm(1, mean=4)
+     y[i] <- "tails"
+   }
+ }
> densityplot( ~x, par.settings =list( plot.symbol=list(col=as.factor(y))))
```



```
> mu_1 <- 0
> mu_2 <- 1
> tau_1 <- 0.5
> tau_2 <- 0.5
> for( i in 1:30 ) {
+   ## Given the observed data, as well as the distribution parameters,
+   ## what are the latent variables?
+   T_1 <- tau_1 * dnorm( x, mu_1 )
+   T_2 <- tau_2 * dnorm( x, mu_2 )
+   P_1 <- T_1 / (T_1 + T_2)
+   P_2 <- T_2 / (T_1 + T_2) ## note: P_2 = 1 - P_1
+   tau_1 <- mean(P_1)
+   tau_2 <- mean(P_2)
+   ## Given the observed data, as well as the latent variables,
+   ## what are the population parameters?
+   mu_1 <- sum( P_1 * x ) / sum(P_1)
+   mu_2 <- sum( P_2 * x ) / sum(P_2)
+   ## print the current estimates
+   print( c(mu_1, mu_2, mean(P_1)) )
+ }
[1] 1.0835357 3.6048714 0.1320495
[1] 0.6797230 3.8663167 0.1865272
```

```
[1] 0.7320122 3.9306341 0.2059336
[1] 0.7910984 3.9574819 0.2165093
[1] 0.8298998 3.9730967 0.2230743
[1] 0.8545108 3.9827182 0.2272189
[1] 0.8701122 3.9887344 0.2298464
[1] 0.8800221 3.9925240 0.2315159
[1] 0.8863270 3.9949222 0.2325783
[1] 0.8903429 3.9964445 0.2332551
[1] 0.8929026 3.9974127 0.2336866
[1] 0.8945350 3.9980293 0.2339618
[1] 0.8955764 3.9984223 0.2341373
[1] 0.8962408 3.9986729 0.2342493
[1] 0.8966648 3.9988327 0.2343208
[1] 0.8969354 3.9989347 0.2343664
[1] 0.8971081 3.9989998 0.2343955
[1] 0.8972184 3.9990414 0.2344141
[1] 0.8972887 3.9990679 0.2344260
[1] 0.8973336 3.9990848 0.2344335
[1] 0.8973623 3.9990956 0.2344384
[1] 0.8973806 3.9991025 0.2344414
[1] 0.8973922 3.9991069 0.2344434
[1] 0.8973997 3.9991097 0.2344447
[1] 0.8974045 3.9991115 0.2344455
[1] 0.8974075 3.9991126 0.2344460
[1] 0.8974094 3.9991134 0.2344463
[1] 0.8974107 3.9991138 0.2344465
[1] 0.8974115 3.9991141 0.2344466
[1] 0.8974120 3.9991143 0.2344467
> myEM <- normalmixEM( x, mu = c(0,1), sigma=c(1,1), sd.constr=c(1,1) )
number of iterations= 21
> ## number of iterations= 21
> myEM$mu ## the means of the two distributions
[1] 0.8974058 3.9991120
> myEM$lambda ## the mixing probabilities
[1] 0.2344461 0.7655539
```