R 로 배우는 데이터 과학

1 장 없음

2 장 데이터 과학으로 풍덩

2 장 01 절

(1)

> 3.141592*25.5*25.5 또는 > pi*25.5*25.5

2 장 02 절

(1) 생략

(2)

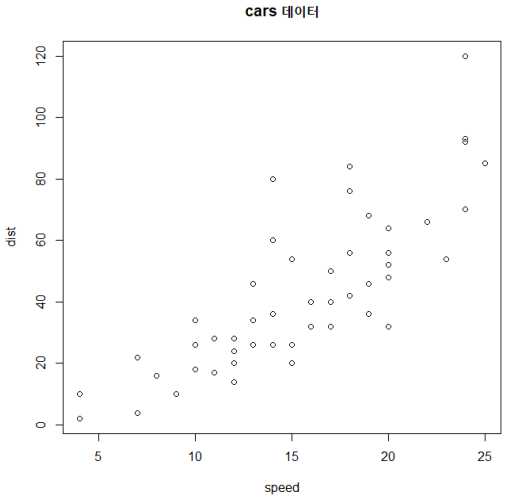
Sepal.Length: 꽃받침의 길이
Sepal.Width: 꽃받침의 너비
Petal.Length: 꽃잎의 길이
Petal.Width: 꽃잎의 너비
Species: 품종 (Setosa, Versicolor, Virginica 중의 하나)

2 장 03 절

(1)

(2)

> plot(cars, main="cars 데이터")



2 장 04 절 없음

2 장 05 절

(1)

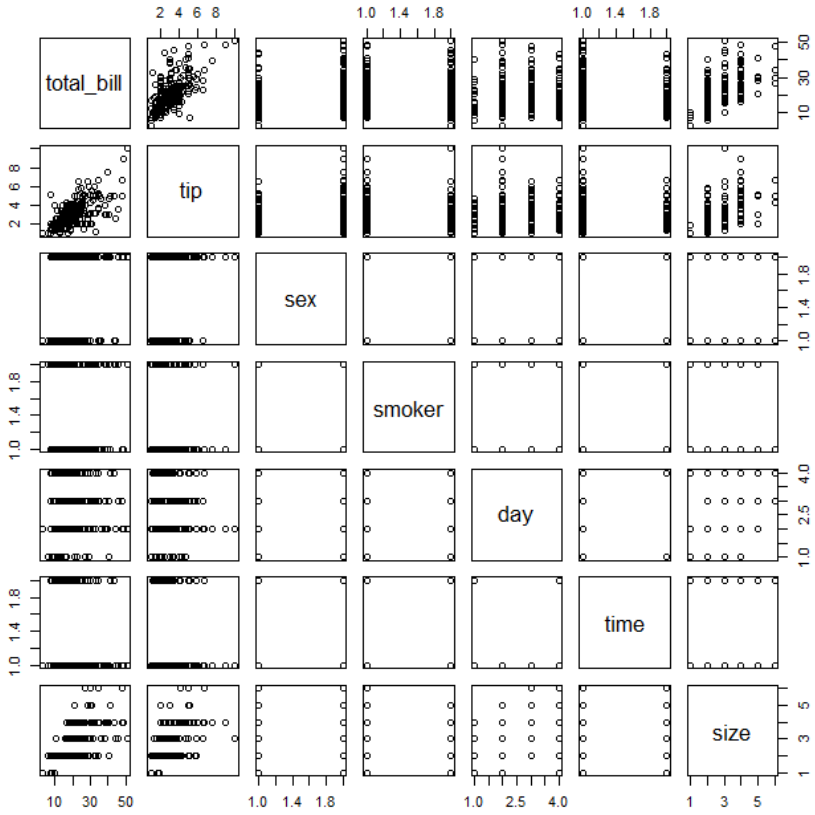
```
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa     :50
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

Sepal.Length 속성은 최소가 4.3, 최대가 7.9 이고, 1/4 선은 5.1, 중앙값은 5.8, 평균은 5.843,

3/4 선은 6.4 이다. 다른 세 속성도 비슷하게 해석하면 된다. Species 속성은 setosa, versicolor, virginica 품종에 속하는 샘플이 각각 50 개씩임을 알려준다.

(2)

> plot(tips)



total_bill 이 클수록 tip 이 크다. size 가 클수록 total_bill 이 크다. size 가 클수록 tip 이 크다. 이 그래프에서는 다른 속성 쌍의 상관관계는 불분명하다.

2 장 06 절 생략

3 장 R 의 데이터형과 연산

3 장 01 절 없음

3 장 02 절

(1)

```
x = 1
y = 2
temp = y
y = x
x = temp
x
y
```

3 장 03 절

(1)

```
x = 0
y = 0
x/y

x = Inf
y = Inf
x/y
```

(2)

```
1 < 2    # 비교 연산의 결과 TRUE 가 출력된다.
4 < 3    # 비교 연산의 결과 FALSE 가 출력된다.
# 즉, 비교 연산의 결과는 참과 거짓으로 구분되어 출력된다.
#다만, NA 와의 비교는 NA 가 출력되므로 유의한다.
```

(3)

```
blood.type1 = c("A", "B", "O", "AB")
blood.type2 = factor(c("A", "B", "O", "AB"))
blood.type1
blood.type2
is.character(blood.type1)
is.character(blood.type2)
is.factor(blood.type1)
is.factor(blood.type2)
# blood.type1 은 단순히 문자를 벡터로 묶음. blood.type2 는 레벨이 구분되어 분류에 용이함.
```

3 장 04 절

(1)

```
x = 3
1 < x & x < 5 # x 가 3 인 경우에는 TRUE 가 반환됨
x = 0
1 < x & x < 5 # x 가 0 인 경우에는 FALSE 가 반환됨
```

(2)

x%%3!=0 | x%%4!=0 # 조건식 !(x%%3==0 & x%%4==0)과 같은 의미임.

3 장 05 절

(1)

```
x = c(1:5)
x = c(x, c(6:10))
x
```

(2)

```
x = c(1:10)
x = x[seq(2,length(x),2)]
x
```

3 장 06 절

(1)

```
plot(Titanic)
# 수행 결과, 1 등석과 2 등석의 남녀 어린이의 생존율이 높음을 알 수 있다.
Titanic
# 데이터를 면밀히 살펴보면, 1 등석에서는 남자어린이 5 명, 여자어린이 1 명이 모두 생존,
# 2 등석에서는 남자어린이 11 명, 여자어린이 13 명이 모두 생존했음을 알 수 있다.
```

(2)

```
x = array(1:24, c(4,6))
x = x[,seq(1,dim(x)[2],2)] # dim(x)수행 결과의 2 번째 항목이 열의 개수임.
# x = x[,seq(1,ncol(x),2)] # ncol 은 열의 개수를 출력하므로 이렇게 대체해도 됨.
```

3 장 07 절

(1)

```
mean(na.omit(airquality$Ozone))
```

(2)

```
name = c("철수", "춘향", "길동")
age = c(22, 20, 25)
gender = factor(c("M", "F", "M"))
blood.type = factor(c("A", "O", "B"))
patients = data.frame(name, age, gender, blood.type)
str(patients) # name 은 범주형으로 표현됨. 분류하기 용이하도록 자동으로 범주형으로 표시함.
```

3 장 08 절

(1)

```
patients = data.frame(name = c("철수", "춘향", "길동"), age = c(22, 20, 25),
                      gender = factor(c("M", "F", "M")), blood.type = factor(c("A", "O",
"B")))
no.patients = data.frame(day = c(1:6), no = c(50, 60, 55, 52, 65, 58))
listPatients = list(patients=patients, no.patients = no.patients)
```

```
listPatients$room = 30
listPatients
```

(2)

```
listPatients$room=NULL
listPatients
```

4 장 데이터 취득과 정제

4 장 01 절

(1)

```
students = read.table("C:/Sources/students.txt", header = T)
students$average = (students$korean + students$english + students$math) / 3
students
write.csv(students, file="C:/Sources/output(연습문제 1).csv", quote=F)
```

4 장 02 절

(1)

```
total = 0
for(i in c(1:10)) {
   if (i%%2 == 0) {
      total=total + i
   }
}
total
```

(2)

```
for(i in seq(1,10,2)){
   print(i)
```

```
}
```

(3)

```
for(i in 2:10) {
  check = 0
  j = 2
  while(j<i) { # 1 과 자기자신으로는 늘 나눠떨어지므로 해당 부분을 생략
    if(i%%j == 0) {
      check = check+1
      break # 나눠 떨어지는 게 하나라도 있으면 break
    }
    j = j+1
  }
  if(check == 0) { # 한번도 나눠떨어지지 않는 수가 소수임
    print(i)
  }
} # 이외에도 다양한 방법이 존재함.
```

4 장 03 절

(1)

```
fact <- function(x) {
  if(x == 1){
    return(1)
  }
```

```
  else if(x > 1){
    return(x * fact(x-1))
  }
}
fact(5) # 5!을 계산해 봄
```

(2)

```
pn <- function(i) {
  check = 0
  for(j in 1:i) {
    if(i%%j == 0) {
      check = check+1
    }
  }
  if(check == 2) {
    return(T)
  }
  else {
    return(F)
  }
}
pn(10) # 10 은 소수가 아님. FALSE 출력
pn(5) # 5 는 소수가 맞음. TRUE 출력
```

4 장 04 절

(1)

```
library(MASS)
table(is.na(Cars93)) # 총 13 개의 NA 가 있음
for(i in c(1:ncol(Cars93))) {
  print(i)
  print(table(is.na(Cars93[,i])))
}
# 수행 결과, 23 번 열에 NA 2 개, 24 번 열에 NA 11 개가 있음
# Rear.seat.room 과 Luggage.room 에 NA 가 존재하는 것임
# ? Cars93 에서 알 수 있듯 Rear.seat.room 은 2-seater vehicles 에서 NA 가 존재하며,
# Luggage.room 은 vans 에서 NA 가 존재함.
# 즉, 2 인승 자동차의 경우 뒷좌석이 없으므로 NA, 밴은 짐 싣는 공간이 따로 구분되어 있지 않으므로 NA 임
```

(2)

```
mean(Cars93$Luggage.room, na.rm=T)*28.316847 # 1 세제곱피트는 28.316847 리터임
```

4 장 05 절

(1)

```
car = cars # 이상값을 NA 로 바꾸기 위해 데이터 복제
boxplot(car$dist)
boxplot(car$dist)$stats
```

```
car$dist = ifelse(car$dist<2 | car$dist>93, NA, car$dist)
mean(car$dist, na.rm=T)
```

(2)

```
chick = ChickWeight # 이상값을 NA 로 바꾸기 위해 데이터 복제
boxplot(chick$weight)
boxplot(chick$weight)$stats
chick$weight = ifelse(chick$weight<35 | chick$weight>309, NA, chick$weight)
mean(chick$weight, na.rm=T) # 평균 118 그램 정도 나옴
```

5 장 데이터 가공

5 장 01 절 없음

5 장 02 절

(1)

```
> gapminder %>% filter(country=="Korea, Rep.") %>% select(pop) %>% max()
[1] 49044790
> gapminder %>% filter(country=="Korea, Rep.") %>% filter(pop==49044790)
# A tibble: 1 x 6
  country     continent  year lifeExp      pop gdpPercap
  <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
1 Korea, Rep. Asia       2007    78.6 49044790    23348.
```

(2)

```
> gapminder %>% filter(year==2007 & continent=="Asia") %>% select(pop) %>% sum()
[1] 3811953827
```

5 장 03 절

(1)

```
> gapminder %>% filter(country == "Korea, Rep.") %>% select( year, country, gdpPercap, lifeExp)
# A tibble: 12 x 4
    year country     gdpPercap lifeExp
   <int> <fct>           <dbl>   <dbl>
 1  1952 Korea, Rep.     1031.    47.5
 2  1957 Korea, Rep.     1488.    52.7
 3  1962 Korea, Rep.     1536.    55.3
 4  1967 Korea, Rep.     2029.    57.7
 5  1972 Korea, Rep.     3031.    62.6
 6  1977 Korea, Rep.     4657.    64.8
 7  1982 Korea, Rep.     5623.    67.1
 8  1987 Korea, Rep.     8533.    69.8
 9  1992 Korea, Rep.    12104.    72.2
10  1997 Korea, Rep.    15994.    74.6
11  2002 Korea, Rep.    19234.    77.0
12  2007 Korea, Rep.    23348.    78.6
> gapminder %>% filter(country == "China") %>% select( year, country, gdpPercap, lifeExp)
# A tibble: 12 x 4
    year country gdpPercap lifeExp
   <int> <fct>       <dbl>   <dbl>
 1  1952 China         400.    44
 2  1957 China         576.    50.5
 3  1962 China         488.    44.5
```

```
 4   1967 China          613.     58.4
 5   1972 China          677.     63.1
 6   1977 China          741.     64.0
 7   1982 China          962.     65.5
 8   1987 China         1379.     67.3
 9   1992 China         1656.     68.7
10   1997 China         2289.     70.4
11   2002 China         3119.     72.0
12   2007 China         4959.     73.0
> gapminder %>% filter(country == "Japan") %>% select( year, country, gdpPercap,
lifeExp)
# A tibble: 12 x 4
      year country gdpPercap lifeExp
     <int> <fct>        <dbl>   <dbl>
 1   1952 Japan        3217.     63.0
 2   1957 Japan        4318.     65.5
 3   1962 Japan        6577.     68.7
 4   1967 Japan        9848.     71.4
 5   1972 Japan       14779.     73.4
 6   1977 Japan       16610.     75.4
 7   1982 Japan       19384.     77.1
 8   1987 Japan       22376.     78.7
 9   1992 Japan       26825.     79.4
10   1997 Japan       28817.     80.7
11   2002 Japan       28605.     82
12   2007 Japan       31656.     82.6
```

(2)

```
> gapminder %>% filter(continent == "Africa") %>% group_by(year) %>%
summarize(s=sum(pop)) -> s1
> gapminder %>% filter(continent == "Europe") %>% group_by(year) %>%
summarize(s=sum(pop)) -> s2
> s1$s > s2$s
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
> s1[s1$s > s2$s, "year"]
# A tibble: 5 x 1
   year
  <int>
1  1987
2  1992
3  1997
4  2002
5  2007
```

(3)

```
> gapminder_unfiltered %>% group_by(country) %>% summarize(n=n()) %>%
filter(n>12) %>%  arrange(desc(n))
# A tibble: 45 x 2
   country                n
   <fct>              <int>
 1 Czech Republic        58
 2 Denmark               58
 3 Finland               58
 4 Iceland               58
```

5	Japan	58
6	Netherlands	58
7	Norway	58
8	Portugal	58
9	Slovak Republic	58
10	Spain	58
11	Sweden	58
12	Switzerland	58
13	Taiwan	58
14	Austria	57
15	Belgium	57
16	Bulgaria	57
17	Canada	57
18	France	57
19	Hungary	57
20	United States	57
21	Australia	56
22	Italy	56
23	New Zealand	55
24	Poland	52
25	Luxembourg	49
26	Latvia	42
27	China	36
28	Slovenia	32
29	Germany	26
30	Russia	20
31	Ukraine	20
32	Belarus	18

33	Estonia	18
34	Lithuania	18
35	Costa Rica	13
36	Cuba	13
37	Greece	13
38	Ireland	13
39	Libya	13
40	Mexico	13
41	Puerto Rico	13
42	Sri Lanka	13
43	Thailand	13
44	Uganda	13
45	United Kingdom	13

5 장 04 절

(1)

p. 172 의 코드 예에서 names(elec_gen) = substr(names(elec_gen), 2, nchar(names(elec_gen))) 명령에 의해 첫번째 열 이름인 "country" 가 "ountry" 가 되는 문제가 발생하였다.
names(elec_gen) 대신 첫번째 원소를 제외한 names(elec_gen)[2: length(names(elec_gen))] 를 사용하면 이런 문제를 방지할 수 있고, 여기서 열 이름의 개수는 length(names(elec_gen)) 을 사용하여 알 수 있다.

(2)

```
> laf = read.csv("literacy_rate_adult_female_percent_of_females_ages_15_above.csv",
header=TRUE, sep=",")
> lam = read.csv("literacy_rate_adult_male_percent_of_males_ages_15_and_above.csv",
header=TRUE, sep=",")
> lat = read.csv("literacy_rate_adult_total_percent_of_people_ages_15_and_above.csv",
header=TRUE, sep=",")
> lyf = read.csv("literacy_rate_youth_female_percent_of_females_ages_15_24.csv",
header=TRUE, sep=",")
> lym = read.csv("literacy_rate_youth_male_percent_of_males_ages_15_24.csv",
header=TRUE, sep=",")
> lyt = read.csv("literacy_rate_youth_total_percent_of_people_ages_15_24.csv",
header=TRUE, sep=",")


> laf_tidy=gather(laf, -country, key="year", value="adult_female")
> lam_tidy=gather(lam, -country, key="year", value="adult_male")
> lat_tidy=gather(lat, -country, key="year", value="adult_total")
```

```
> lyf_tidy=gather(lyf, -country, key="year", value="youth_female")
> lym_tidy=gather(lym, -country, key="year", value="youth_male")
> lyt_tidy=gather(lyt, -country, key="year", value="youth_total")

> literacy = merge(laf_tidy, lam_tidy)
> literacy = merge(literacy, lam_tidy)
> literacy = merge(literacy, lat_tidy)
> literacy = merge(literacy, lyf_tidy)
> literacy = merge(literacy, lym_tidy)
> literacy = merge(literacy, lyt_tidy)
> literacy = na.omit(literacy)
```

주의할 점: 일반적인 데이터프레임의 형태에서는 na.omit 을 사용하여 결측값을 제거한 후 가공을 하는 것이 보통이지만, 이 경우에는 데이터의 구조를 변경한 이후에 na.omit 을 사용해야 한다.

6 장 데이터 시각화

6 장 01 절

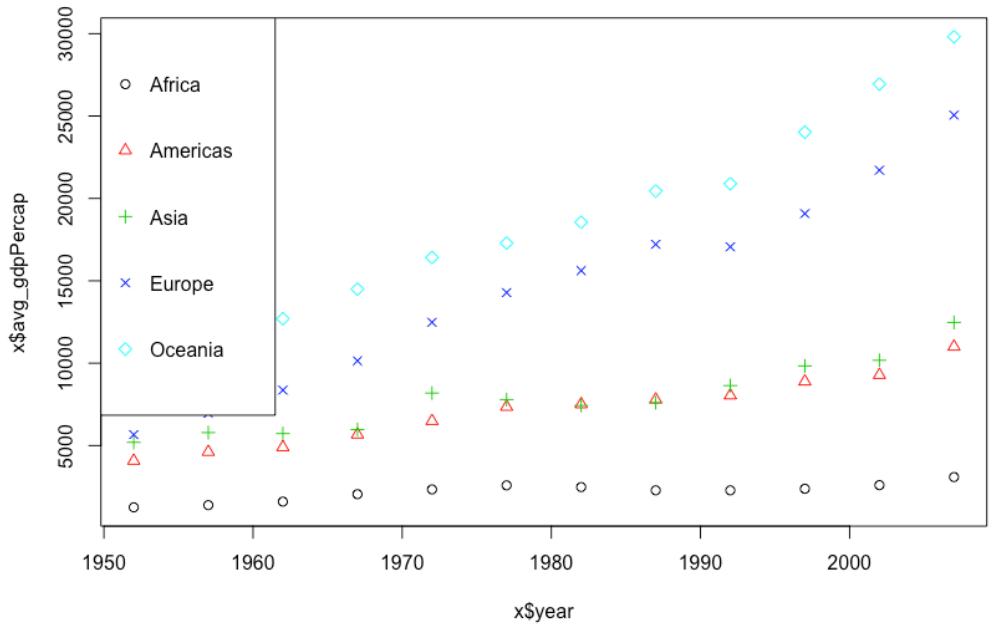
(1)

먼저 데이터 가공을 통해 각 대륙의 연도별 gdpPercap 의 평균치를 구하여 x 라는 변수에 저장한다.

```
> gapminder %>% group_by(year, continent) %>% summarize(avg_gdpPercap = mean(gdpPercap)) -> x
```

plot 와 legend 함수를 이용해 연도별 변화를 플롯한다.

```
> plot(x$year, x$avg_gdpPercap, col=x$continent, pch=c(1:length(levels(x$continent))))
> legend("topleft", legend=levels(x$continent), col=c(1:length(levels(x$continent))), pch=c(1:length(levels(x$continent))))
```



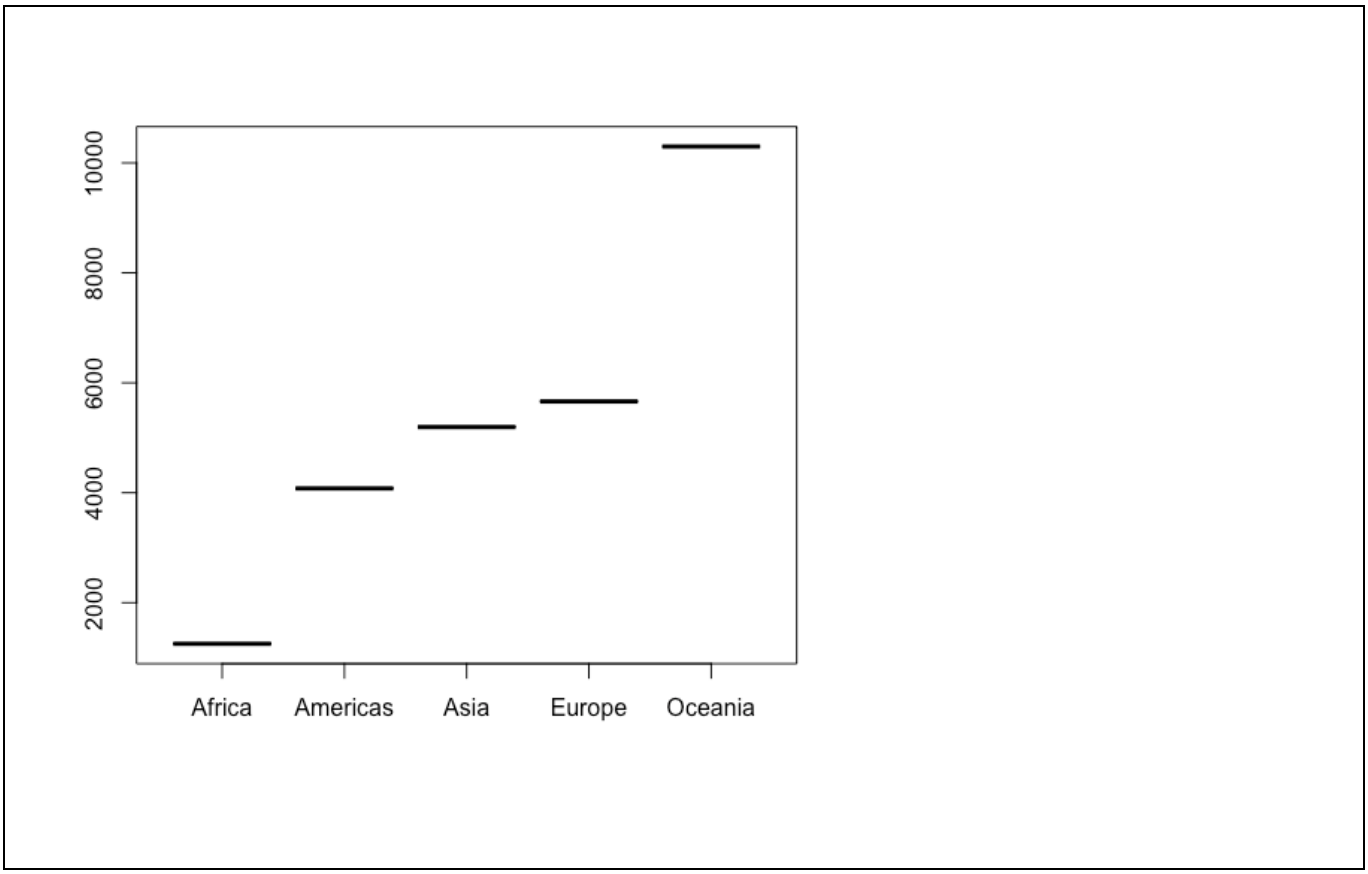
(2)

먼저 데이터 가공을 통해 1952년의 gdpPercap과 lifeExp의 대륙별 평균을 추출한 후 x 라는 변수에 저장한다.

```
> gapminder %>% filter(year==1952) %>% group_by(continent) %>%
summarise(m=mean(gdpPercap)) -> x
> x
# A tibble: 5 x 2
  continent        m
  <fct>        <dbl>
1 Africa       1253.
2 Americas     4079.
3 Asia         5195.
4 Europe       5661.
5 Oceania     10298.
```

plot 함수를 이용해 시각화한다.

```
> plot(x$continent, x$m)
```


2000
4000
6000
8000
10000
Africa
Americas
Asia
Europe
Oceania

6 장 02 절

(1)

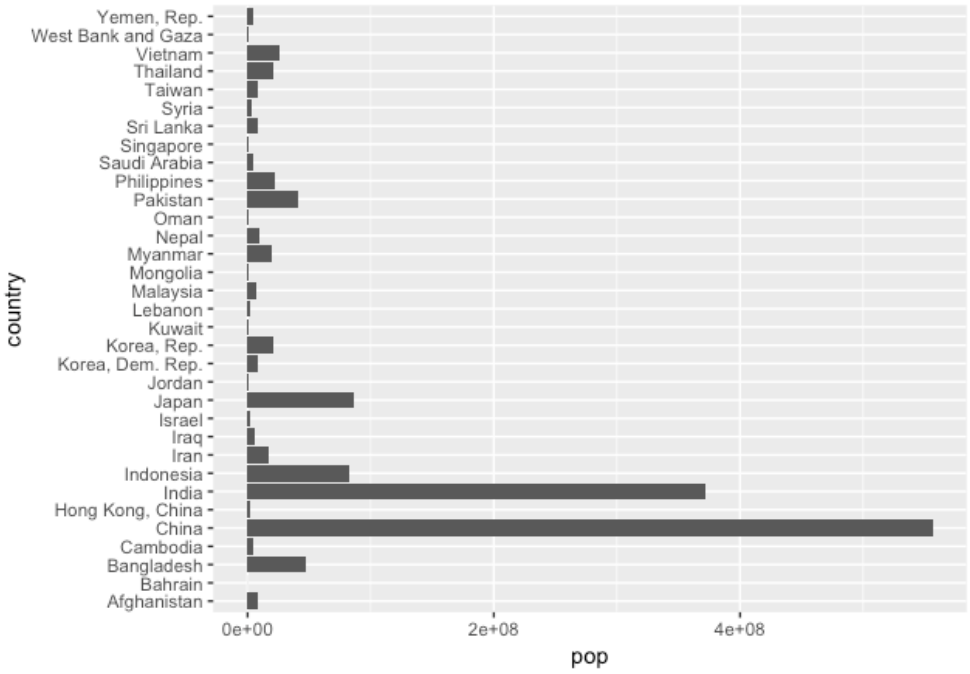
그림 6-13 은 다음 명령어에 의한 시각화이다.

>gapminder%>%filter(year==1952&continent=="Asia")%>%ggplot(aes(reorder(coun try, pop), pop))+geom_bar(stat="identity")+coord_flip()

여기서 reorder(country, pop) 은 country 를 pop 의 크기에 따라 재배열하라는 의미이므로, 그림 6-13 과 같이 그래프의 x 축에 pop 의 크기 순으로 정렬된 country 가 지정되게 된다.

그에 비해, 주어진 명령어는 country 를 재배열하지 않는 시각화이므로 다음과 같은 시각화 결과를 얻게 된다.

> gapminder %>% filter(year == 1952 & continent == "Asia") %>% ggplot(aes(country, pop))+geom_bar(stat="identity")+coord_flip()

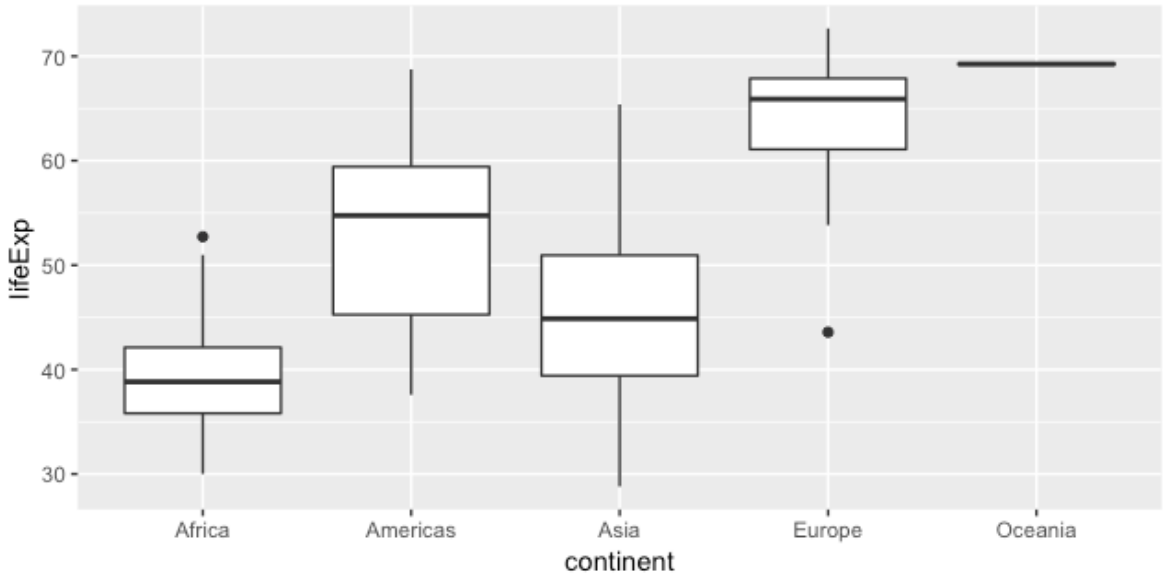


(2)

그림 6-19 는 다음 명령어에 의한 시각화이다 .

x = filter(gapminder, year == 1952)

x %>% ggplot(aes(continent, lifeExp)) + geom_boxplot()



주어진 명령어와 시각화 결과는 아래와 같다.

ggplot(gapminder, aes(continent, lifeExp)) + geom_point(alpha=0.2, size= 1.0, position="jitter")

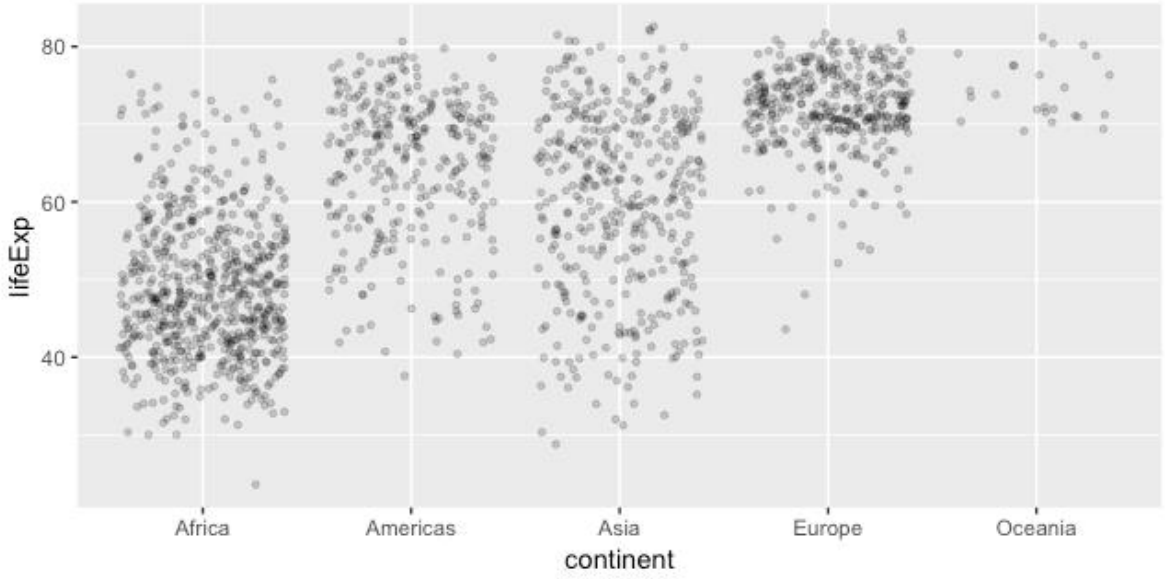
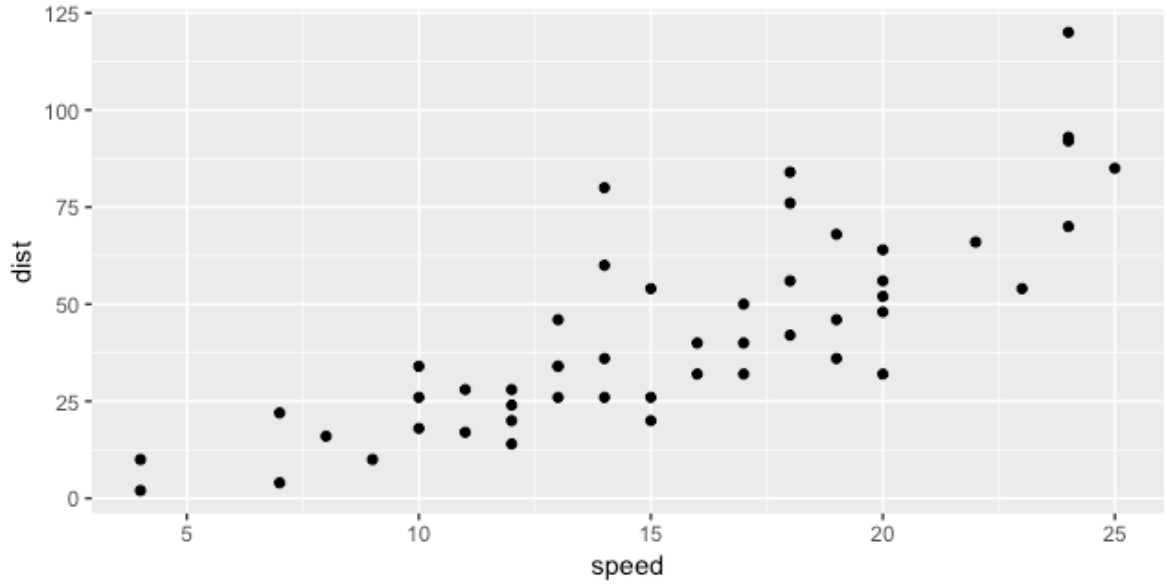


그림 6-19 가 1952년도의 대륙별 기대수명 데이터를 통계적으로 요약한 관점에서 시각화한 결과라면,
주어진 명령어는 같은 데이터를 시각화하되, 개별 데이터의 분포를 사실적으로 확인할 수 있도록 보여준 결과라고 할 수 있다.

6 장 03 절

(1)

```
> cars %>% ggplot(aes(speed, dist)) + geom_point()
```



(2)

주의할 점: 데이터의 구조 변경이 먼저 이루어져야 ggplot 에 의해 그래프를 그릴 수 있다. (ggplot 에서 멀티플롯을 그리기는 의외로 번거롭다)

우선 그림 6-27 과 같이 가로축에 iris 샘플을 지정하기 위해 id 라는 열을 새로 추가한다.

```
> id = 1:150
> iris1 = cbind(id, iris1)
```

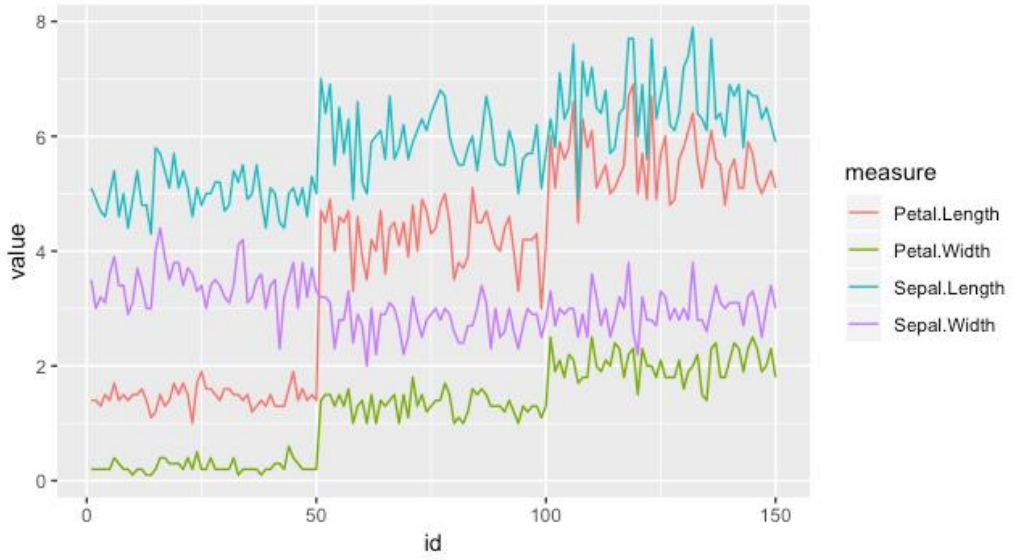
그 다음 샘플 1개당 존재하는 4개의 관측 변수를 하나의 행으로 분리하여 데이터의 구조를 변경한다.

```
> iris_tidy = gather(iris1, -id, -Species, key="measure", value="value")
Species        measure value
1         setosa Sepal.Length     5.1
2         setosa Sepal.Length     4.9
```

3 setosa Sepal.Length 4.7
4 setosa Sepal.Length 4.6
5 setosa Sepal.Length 5.0
6 setosa Sepal.Length 5.4
.....

이제 ggplot 을 이용하여 x축에 id 를, y축에 관측치를 지정하고, 4개의 변수를 col 로 구분하여 별도의 선그래프로 그린다.

```
> iris_tidy %>% ggplot(aes(id, value, col=measure)) + geom_line()
```



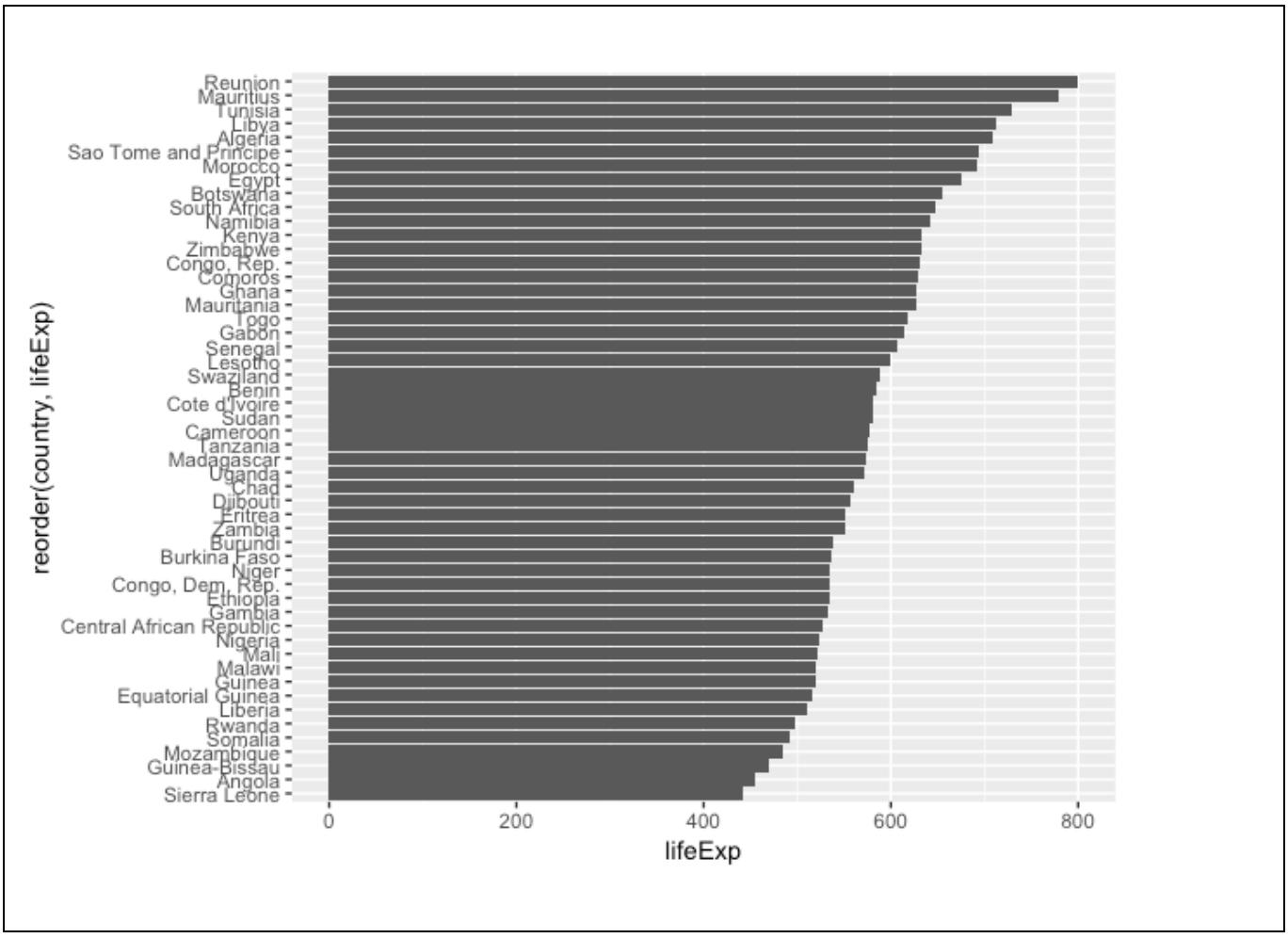
(3)

그림 6-359(b) 는 아래의 명령어에 의한 시각화 결과이다.

```
> gapminder %>% filter(continent == "Africa") %>% ggplot(aes(country, lifeExp)) +
geom_bar(stat = "identity") + coord_flip()
```

x축 (coord_flip 함수에 의해 수직축으로 표시되었다) 에 지정된 country 를 lifeExp 의 크기순으로 재배열해야 하므로 다음과 같이 reorder 함수를 이용한다.

```
> gapminder %>% filter(continent == "Africa") %>% ggplot(aes(reorder(country, lifeExp),
lifeExp)) + geom_bar(stat = "identity") + coord_flip()
```


reorder(country, lifeExp)
Reunion
Mauritius
Tunisia
Libya
Algeria
Sao Tome and Principe
Morocco
Egypt
Botswana
South Africa
Namibia
Kenya
Zimbabwe
Congo, Rep.
Comoros
Ghana
Mauritania
Togo
Gabon
Senegal
Lesotho
Swaziland
Benin
Cote d'Ivoire
Sudan
Cameroon
Tanzania
Madagascar
Uganda
Chad
Djibouti
Eritrea
Zambia
Burundi
Burkina Faso
Niger
Congo, Dem. Rep.
Ethiopia
Gambia
Central African Republic
Nigeria
Mali
Malawi
Guinea
Equatorial Guinea
Liberia
Rwanda
Somalia
Mozambique
Guinea-Bissau
Angola
Sierra Leone
0
200
400
600
800
lifeExp

6 장 04 절

(1)

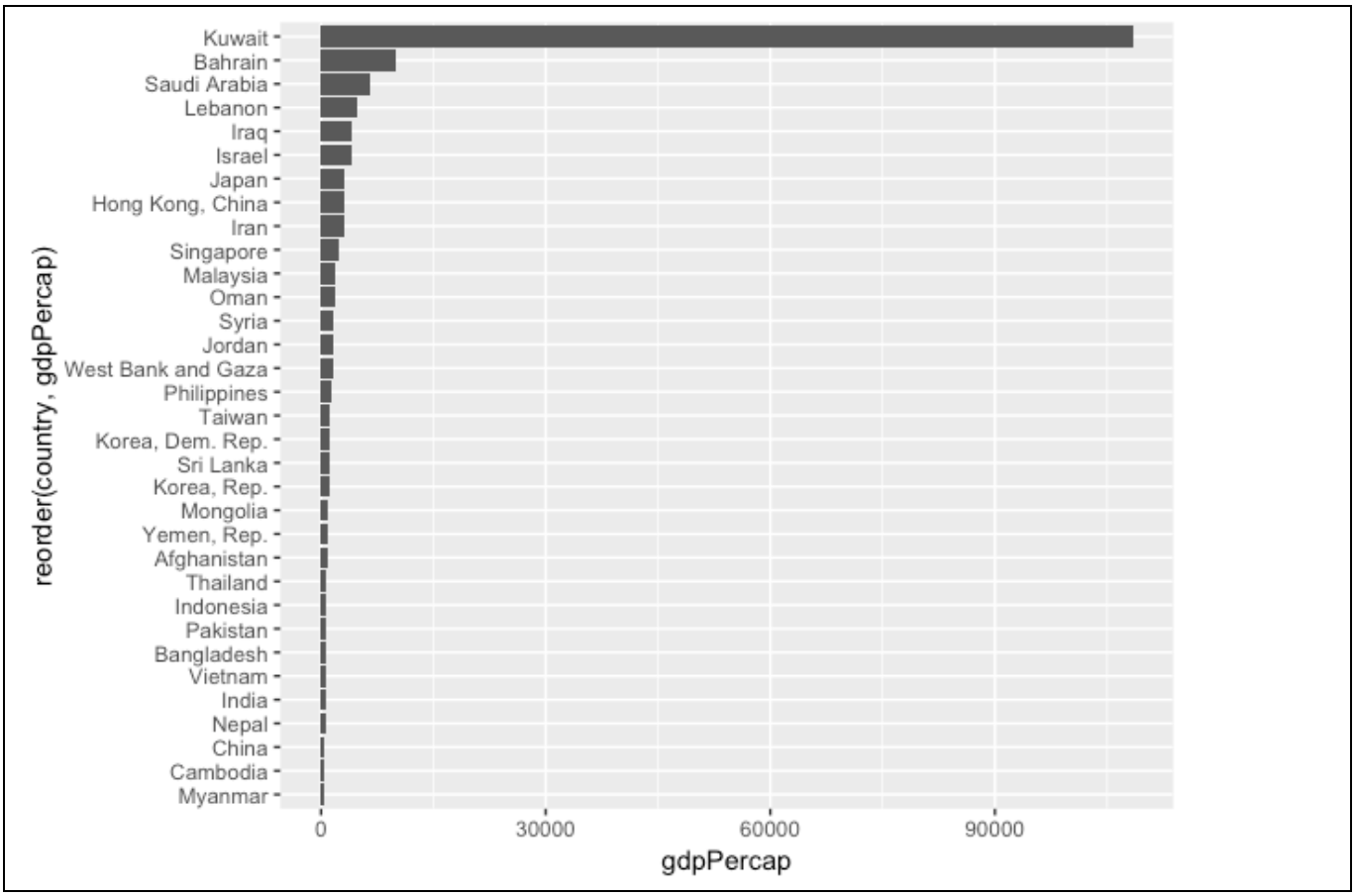
```
gapminder%>%filter(country=="Kuwait"|country=="Saudi Arabia"|country==
"Iraq"|country=="Iran"|country=="Korea, Rep."|country=="China"|country== "Japan") %>%
ggplot(aes(year, gdpPercap, col = country)) + geom_point() + geom_ line()
```

%in% 는 특수연산자로 벡터 내 특정 값 포함 여부를 확인할 때 사용할 수 있는 연산자이다. %in% 를 이용해 위의 명령어의 논리식 부분을 간략히 표현할 수 있다.

```
gapminder%>%filter(country %in% c("Kuwait", "Saudi Arabia",  "Iraq", "Iran", "Korea, Rep.",
"China"," Japan") %>% ggplot(aes(year, gdpPercap, col = country)) + geom_point() +
geom_ line()
```

(2)

```
> gapminder %>% filter(continent=="Asia"&year==1952) %>%
ggplot(aes(reorder(country, gdpPercap), gdpPercap)) + geom_bar(stat="identity") +
coord_flip()
```


Kuwait
Bahrain
Saudi Arabia
Lebanon
Iraq
Israel
Japan
Hong Kong, China
Iran
Singapore
Malaysia
Oman
Syria
Jordan
West Bank and Gaza
Philippines
Taiwan
Korea, Dem. Rep.
Sri Lanka
Korea, Rep.
Mongolia
Yemen, Rep.
Afghanistan
Thailand
Indonesia
Pakistan
Bangladesh
Vietnam
India
Nepal
China
Cambodia
Myanmar
reorder(country, gdpPercap)
0
30000
60000
90000
gdpPercap

7 장 모델링과 예측 : 선형 회귀

7 장 01 절

(1)

$\alpha_0 = 2500000, \alpha_1 = 500000$

(2) 생략

7 장 02 절

(1)

x_i	3.0	6.0	9.0	12.0
예측값 $f(x_i)$	2.9	3.8	4.7	5.6
그라운드트루스 y_i	3.0	4.0	5.5	6.5
오차	0.1	0.2	0.8	0.9

평균 제곱 오차=0.375

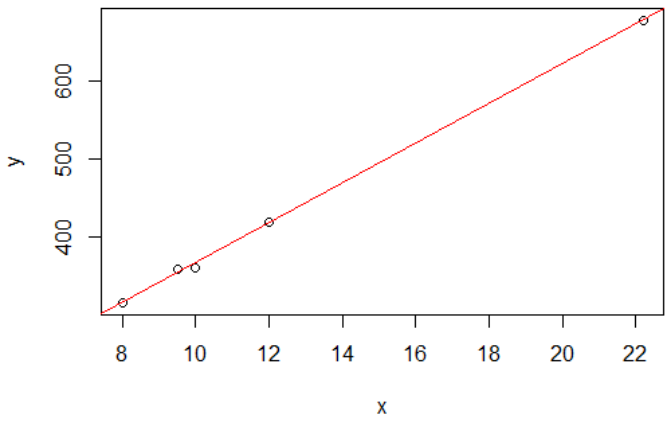
(2)

그림 7-3(a)의 평균 절댓값 오차=0.25

그림 7-3(b)의 평균 절댓값 오차=0.125

7 장 03 절

(1)



x_i	10.0	12.0	9.5	22.2	8.0
예측값 $f(x_i)$	366.9	418.1	354.1	679.2	315.7
그라운드트루스 y_i	360.2	420.0	359.5	679.0	315.3
오차	-6.7	1.9	5.4	-0.2	-0.4

잔차 제곱합=77.864

평균 제곱 오차=15.573

예측 결과=(379.7, 750.8,494.9)

(2)

```
nx=data.frame(x=c(1.2,2.0,20.65))
ny=predict(m,newdata=nx)
plot(nx$x, ny, col='red', pch=20)
abline(m)
```

7 장 04 절

(1)

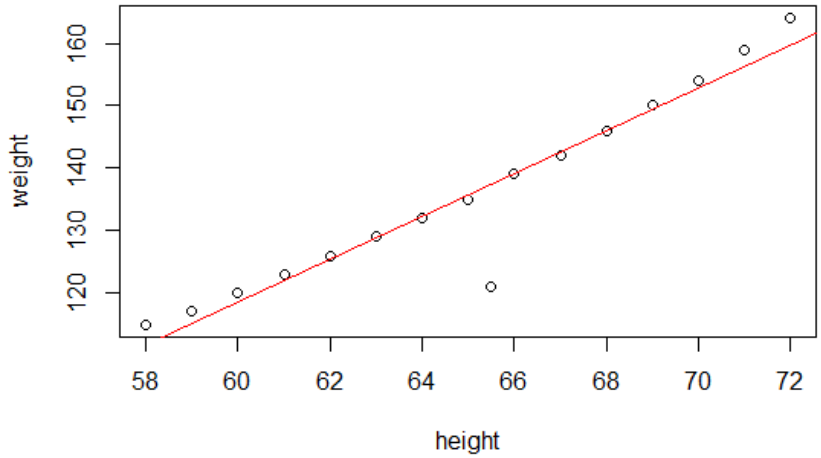
```
> m=lm(weight~height,data=women)
> newd=data.frame(height=c(130,140,151))
> predict(m,newdata=newd)
       1        2        3
360.9833 395.4833 433.4333
```

(2) 생략

7 장 05 절

(1)

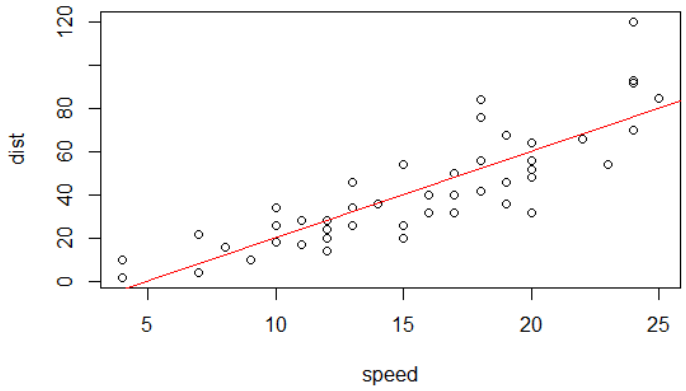
```
> w=rbind(women,c(65.5,121)) # 샘플 추가
> m=lm(weight~height,data=w)
> plot(w)
> abline(m,col='red')
> summary(m)

Call:
lm(formula = weight ~ height, data = w)

Residuals:
    Min      1Q  Median      3Q     Max
-16.3535  -0.1035   0.6465   1.5661   4.4113

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -86.7087    18.4867   -4.69 0.000347 ***
height        3.4208     0.2837   12.06 8.79e-09 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 4.749 on 14 degrees of freedom
Multiple R-squared:  0.9122,    Adjusted R-squared:  0.9059
F-statistic: 145.4 on 1 and 14 DF,  p-value: 8.788e-09
```



height 변수의 계수의 p-값이 8.79e-09 로서 원래 women 데이터보다 높아졌다. 그 이유는 이상치로 간주할 수 있는 샘플이 하나 추가되어 모델의 오차가 커졌기 때문이다. 하지만 여전히 0.05 보다 작기 때문에 통계적으로 유의미한 모델링이 되었음을 알 수 있다.

(2)

```
> cars1=cars[-c(20,22,23),]
> m=lm(dist~speed,data=cars1)
> plot(cars1)
> abline(m,col='red')
> summary(m)

Call:
lm(formula = dist ~ speed, data = cars1)

Residuals:
    Min      1Q  Median      3Q     Max
-28.193  -8.243  -2.175   5.689  43.843

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -19.6253     6.2242  -3.153  0.00288 **
speed         3.9909     0.3795  10.515 1.05e-13 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 14.02 on 45 degrees of freedom
Multiple R-squared:  0.7107,    Adjusted R-squared:  0.7043
F-statistic: 110.6 on 1 and 45 DF,  p-value: 1.053e-13
```

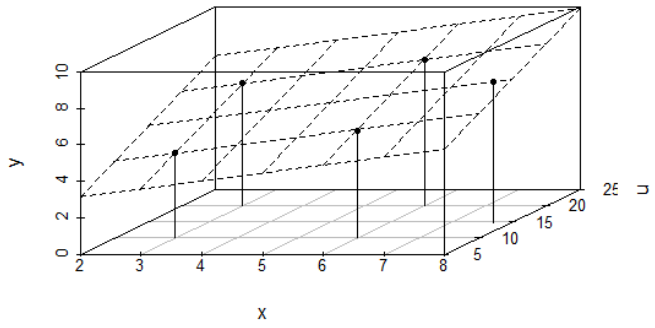


speed 변수의 계수의 p-값이 1.05e-13 으로서 원래 데이터의 1.49e-12 보다 작아졌다. 결과적으로 통계적인 유의미성이 커졌다. 이러한 결과는 speed 값이 14 인 4 개 샘플 중에서 모델의 예측값을 벗어날 것으로 보이는 3 개의 샘플을 제거함으로써 불확실성을 줄였기 때문에 얻어졌다.

7 장 06 절

(1)

```
> library(scatterplot3d)
> x = c(3.0, 6.0, 3.0, 6.0, 7.7)
> u = c(10.0, 10.0, 20.0, 20.0, 14.8)
> y = c(4.65, 5.9, 6.7, 8.02, 7.7)
> scatterplot3d(x, u, y, xlim = 2:8, ylim = 7:23, zlim = 0:10, pch =
20, type = 'h')
> m = lm(y ~ x + u)
> s = scatterplot3d(x, u, y, xlim = 2:8, ylim = 7:23, zlim = 0:10,
pch = 20, type = 'h')
> s$plane3d(m)
> nx = c(7.5, 5.0)
> nu = c(15.0, 12.0)
> new_data = data.frame(x = nx, u = nu)
> ny = predict(m, new_data)
> ny
       1        2
7.632014 5.915890
```



7 장 07 절

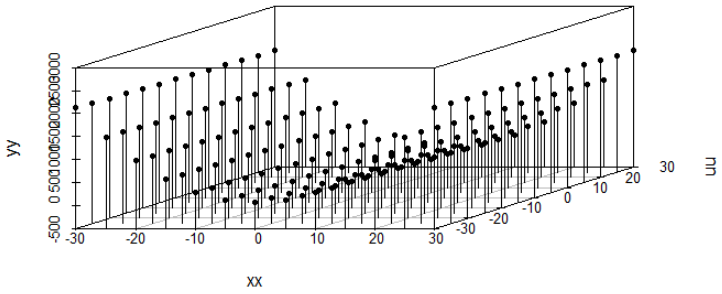
(1)

```
> library(scatterplot3d)
> x = c(3.0, 6.0, 3.0, 6.0, 7.5, 7.5, 15.0)
> u = c(10.0, 10.0, 20.0, 20.0, 5.0, 10.0, 12.0)
> y = c(4.65, 5.9, 6.7, 8.02, 7.7, 8.1, 6.1)
> scatterplot3d(x, u, y, xlim = 2:16, ylim = 4:21, zlim = 0:10, pch
= 20, type = 'h')
> m = lm(y ~ x + u)
> s = scatterplot3d(x, u, y, xlim = 2:16, ylim = 4:21, zlim = 0:10,
pch = 20, type = 'h')
> s$plane3d(m)
> nx = c(7.5, 5.0)
> nu = c(15.0, 12.0)
> new_data = data.frame(x = nx, u = nu)
> ny = predict(m, new_data)
> ny
       1        2
6.888922 6.613455
```

(2)

```
> library(scatterplot3d)
> library(matlab)
> x=seq(-30,30,5)
> u=seq(-30,30,5)
> m=meshgrid(x,u)  # x,u is renamed x,y
> xx=as.vector(m$x)
```

```
> uu=as.vector(m$y)
> y=2.3*xx^2-1.5*uu+10.5
> yy=as.vector(y)
> s=scatterplot3d(xx,uu,yy,xlim=-30:30,ylim=-30:30,zlim=-
300:3000,pch=20,type='h')
```

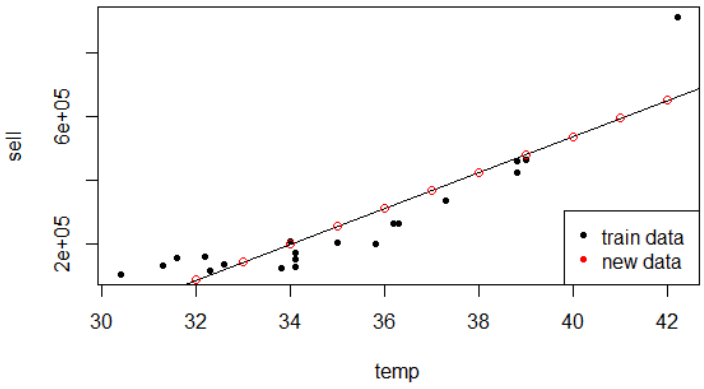


8 장 일반화 선형 모델

8 장 01 절

(1)

```
>
sell=c(423000,207900,464600,460000,264500,107500,161600,131200,20600
0,910400,338600,138300,157400,172100,153000,127200,200600,116100,265
200,132500)
>
temp=c(38.8,34.0,39.0,38.8,36.2,30.4,32.2,34.1,35.0,42.2,37.3,32.6,3
1.6,34.1,34.1,33.8,35.8,32.3,36.3,31.3)
> icecream_sell=data.frame(temp,sell)
> plot(icecream_sell,pch=20,cex=1)
> m=lm(sell~temp,data=icecream_sell)
> newd=data.frame(temp=30:45)
> p=predict(m,newdata=newd)
>
> plot(icecream_sell,pch=20)
> abline(m)
> res=data.frame(temp=newd,sell=p)
> points(res,col='red')
> legend('bottomright',legend=c('train data','new
data'),pch=c(20,20),cex=1,col=c('black','red'))
```



8 장 02 절

(1)

```
> haberman = read.csv("https://archive.ics.uci.edu/ml/machine-
learning-databases/haberman/haberman.data")
> str(haberman)
'data.frame':       305 obs. of  4 variables:
 $ X30 : int  30 30 31 31 33 33 34 34 34 34 ...
 $ X64 : int  62 65 59 65 58 60 59 66 58 60 ...
 $ X1  : int  3 0 2 4 10 0 0 9 30 1 ...
 $ X1.1: int  1 1 1 1 1 1 2 2 1 1 ...
```

첫 번째 샘플이 30, 64, 1, 1 인데 이 값이 변수 이름을 만드는데 사용되어 변수 이름이 X30, X64, X1, X1.1 이 되었다. 그리고 원래 306 개 샘플이었는데 하나가 줄어 305 개가 되었다.

(2)

헬프 명령어 ?predict.glm 을 실행해보면 다음과 같다. 즉 type 에 link, response, terms 세 가지가 가능하다.

```
predict(object, newdata = NULL,
            type = c("link", "response", "terms"),
            se.fit = FALSE, dispersion = NULL, terms = NULL,
            na.action = na.pass, ...)
```

세가지 옵션을 실행한 결과는 다음과 같다.

```
> predict(h, newdata = new_patients, type = 'response')
        1         2
0.2225961 0.8448620
> predict(h, newdata = new_patients, type = 'link')
        1         2
-1.250601  1.694858
> predict(h, newdata = new_patients, type = 'terms')
         age    op_year   no_nodes
1 -0.3075945 0.04748050 0.08613022
2  0.2694866 0.02791278 2.47407600
attr(,"constant")
[1] -1.076617
```

response 는 272 쪽의 결과처럼 확률을 출력한다. link 는 274 쪽의 식(1)의 잠복 변수 /값을

나타낸다. terms 는 모델의 각 항목(변수)의 값을 출력한다.

8 장 03 절

(1) 생략

(2)

지수(exponential): $-\frac{1}{l}$

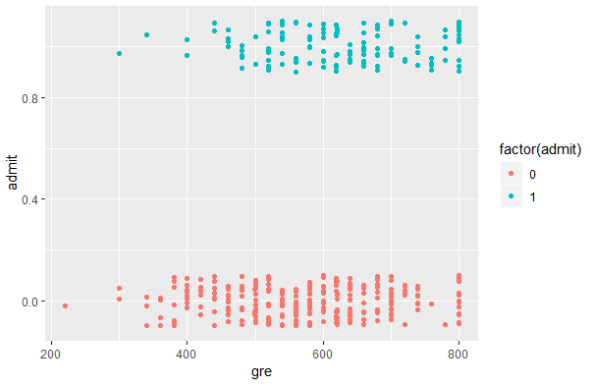
포와송: e^{l}

*** Wikipedia "generalized linear model" 참조

8 장 04 절

(1)

width=0.0



width=0.1

왼쪽은 width=0.0 이고, 오른쪽은 width=0.1 이다. 둘 간에 큰 차이는 없어 보인다.

(2)

```
> ucla = read.csv('https://stats.idre.ucla.edu/stat/data/binary.csv')
> m = glm(admit~., data = ucla, family = binomial)
> s = data.frame(gre = c(376), gpa = c(3.6), rank = c(1))
> predict(m, newdata = s, type ='response')
        1
0.4134304
```

```
> s = data.frame(gre = c(376), gpa = c(3.6), rank = c(2))
> predict(m, newdata = s, type ='response')
        1
0.2870339
> s = data.frame(gre = c(376), gpa = c(3.6), rank = c(3))
> predict(m, newdata = s, type ='response')
        1
0.1869631
> s = data.frame(gre = c(376), gpa = c(3.6), rank = c(4))
> predict(m, newdata = s, type ='response')
        1
0.1160996
```

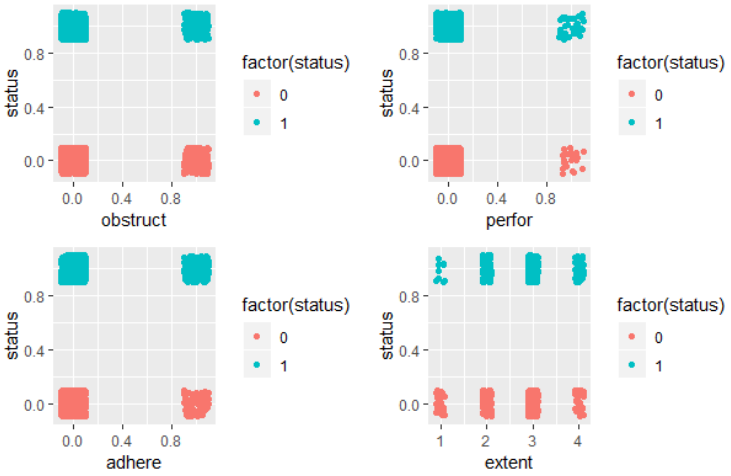
이것을 for 문을 사용하여 다시 작성하면 다음과 같다.

```
> for (r in 1:4) {
+    s = data.frame(gre = c(376), gpa = c(3.6), rank = c(r))
+    predict(m, newdata = s, type ='response')
+ }
```

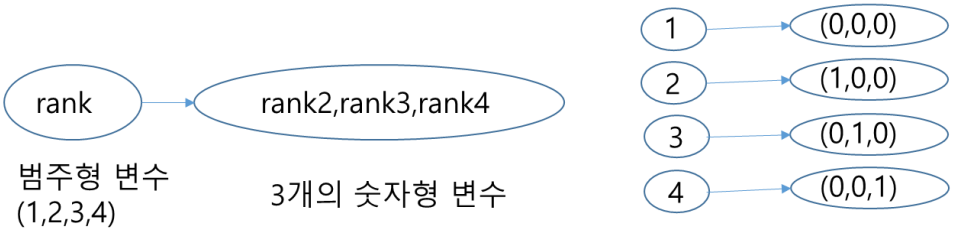
8 장 05 절

(1)

다음은 obstruct, perfor, adhere, extent 에 대해 status 와 상관관계를 그래프로 그린 것이다.



(2)



(3)

```
> ucla = read.csv('https://stats.idre.ucla.edu/stat/data/binary.csv')
> ucla$rank=as.factor(ucla$rank)
> m = glm(admit~., data = ucla, family = binomial)
> coef(m)
 (Intercept)          gre          gpa        rank2        rank3
-3.989979073  0.002264426  0.804037549 -0.675442928 -1.340203916
       rank4
-1.551463677
> deviance(m,type='response')
[1] 458.5175
> s = data.frame(gre = c(376), gpa = c(3.6), rank = as.factor(c(3)))
> predict(m, newdata = s, type ='response')
        1
0.1701981
```

coef(m)의 결과를 보면 설명 변수의 개수가 원래 3 개에서 5 개로 확장된 것을 볼 수 있다. deviance 와 predict 의 결과는 약간의 차이는 있지만 거의 비슷하다.

9 장 분류를 위한 모델

9 장 01 절 생략

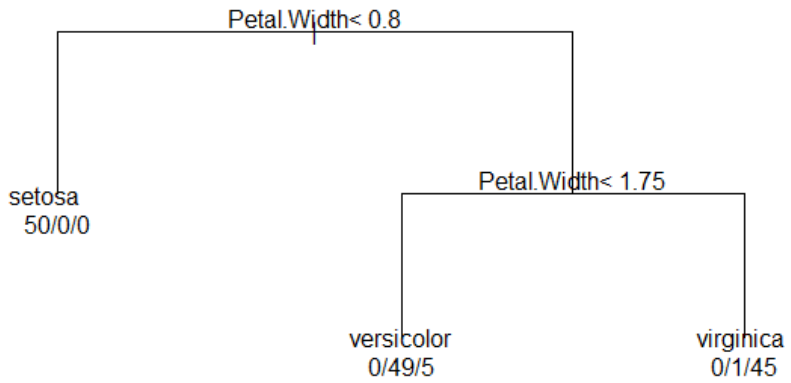
9 장 02 절 생략

9 장 03 절

(1)

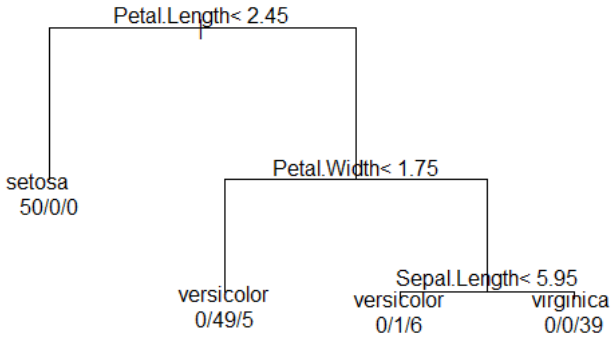
```
> library(rpart)
```

```
> r=rpart(Species~Petal.Width+Sepal.Length+Sepal.Width,data=iris)
> par(mfrow=c(1,1),xpd=NA)
> plot(r)
> text(r,use.n=TRUE)
> p=predict(r,iris,type='class')
> table(p,iris$Species)

p            setosa versicolor virginica
  setosa         50          0         0
  versicolor      0         49         5
  virginica       0          1        45
```



(2)

```
> library(rpart)
> r_prior = rpart(Species~., data = iris, parms = list(prior =
c(0.1, 0.8, 0.1)))
> par(mfrow=c(1,1),xpd=NA)
> plot(r_prior)
> text(r_prior, use.n = TRUE)
```



0/1/6(setosa 0 개, versicolor 1 개, virginica 6 개)을 가진 리프 노드는 viginica 가 가장 많음에도 불구하고 versicolor 의 사전 확률이 0.8 로 가장 높으므로 versicolor 로 분류한다.

(3)

Recursive Partitioning And Regression Trees

9 장 04 절

(1)

rpart.plot(x = stop("no 'x' arg"), type = 2, extra = "auto", under = FALSE, fallen.leaves = TRUE, digits = 2, varlen = 0, faclen = 0, roundint = TRUE, cex = NULL, tweak = 1, clip.facs = FALSE, clip.right.labs = TRUE, snip = FALSE, box.palette = "auto", shadow.col = 0, ...)

(2)

Petal.Length 가 2.5 보다 크거나 같고, Petal.Width 가 1.8 보다 크거나 같기 때문이다.

9 장 05 절

(1)

없다. 램덤포리스트는 보통 수백 개의 트리를 가지는데 이렇게 많은 트리를 일일이 그려주는 것이 어렵기 때문으로 보인다.

(2)

다음 프로그램은 세 가지 옵션, response, prob, vote 를 실행한 결과이다. response 는 1, 2, 3 번 샘플각각에 대해 어떤 부류로 분류했는지를 보여준다. 즉 1 번 샘플은 setosa, 2 번 샘플은 versicolor, 3 번 샘플은 verginica 라고 분류했다고 출력한다. vote 는 득표율을 출력한다. 예를 들어, 2 번 샘플은 setosa 는 0%, versicolor 는 98%, virginica 는 2%를 득표하였다. prob 는 확률을 출력하는데, 득표율을 확률로 간주하므로 vote 와 같은 값을 출력한다.

```
> library(randomForest)
> f=randomForest(Species~.,data=iris)
> newd =
data.frame(Sepal.Length=c(5.11,7.01,6.32),Sepal.Width=c(3.51,3.2,3.3
1),Petal.Length=c(1.4,4.71,6.02),Petal.Width=c(0.19,1.4,2.49))
> predict(f, newdata=newd,type='response')
       1        2         3
    setosa versicolor  virginica
Levels: setosa versicolor virginica
> predict(f, newdata=newd,type='prob')
  setosa versicolor virginica
1      1      0.000     0.000
2      0      0.980     0.020
3      0      0.002     0.998
attr(,"class")
[1] "matrix" "votes"
> predict(f, newdata=newd,type='vote')
  setosa versicolor virginica
1      1      0.000     0.000
2      0      0.980     0.020
3      0      0.002     0.998
attr(,"class")
[1] "matrix" "votes"
```

(3)

```
> library(randomForest)
> f=randomForest(Species~.,data=iris)
```

```
> newd =
data.frame(Sepal.Length=c(4.7,5.31,6.4,5.2,6.3),Sepal.Width=c(3.2,3.
7,3.22,2.71,3.3),Petal.Length=c(1.3,1.5,4.5,3.9,6.1),Petal.Width=c(0
.22,0.2,1.5,1.4,2.5))
> predict(f, newdata=newd,type='prob')
  setosa versicolor virginica
1  1.000      0.000     0.000
2  1.000      0.000     0.000
3  0.000      1.000     0.000
4  0.002      0.994     0.004
5  0.000      0.002     0.998
attr(,"class")
[1] "matrix" "votes"
```

1, 2 번 샘플은 setosa, 3, 4 번 샘플은 versicolor, 4 번 샘플은 virginica 일 확률이 가장 크다.

9 장 06 절

(1)

cost 의 기본값은 1 이다. (https://cran.r-project.org/web/packages/e1071/e1071.pdf 참조)

```
> library(e1071)
> s_default=svm(Species~.,data=iris) # cost 의 기본값은 1
> s_small=svm(Species~.,data=iris,cost=0.1) # cost 를 작게
> s_large=svm(Species~.,data=iris,cost=10) # cost 를 크게
> table(predict(s_default,iris),iris$Species)

            setosa versicolor virginica
  setosa        50         0         0
  versicolor     0        48         2
  virginica      0         2        48
> table(predict(s_small,iris),iris$Species)

            setosa versicolor virginica
  setosa        50         0         0
  versicolor     0        47         5
  virginica      0         3        45
> table(predict(s_large,iris),iris$Species)

            setosa versicolor virginica
  setosa        50         0         0
  versicolor     0        48         0
  virginica      0         2        50
```

기본값 1, 기본값보다 10 배 작은 0.1, 기본값보다 10 배 큰 10 을 실험한 결과이다. cost 를 작게 한 경우에는 훈련 집합에 대한 오류율이 커졌다. 하지만 일반화 능력이 뛰어나 새로운 샘플에 대한 일반화 능력이 클 것으로 예상된다. cost 를 크게 한 경우에는 훈련 집합에 대한 오류율이 작아졌다. 하지만 새로운 샘플에 대한 일반화 능력이 작을 것으로 예상된다.

(2)

총 238 개의 모델을 제공한다. (getModelInfo 함수는 caret 라이브러리가 제공한다.)

```
> names(getModelInfo())
  [1] "ada"                "AdaBag"             "AdaBoost.M1"
  [4] "adaboost"           "amdai"              "ANFIS"
  [7] "avNNet"             "awnb"               "awtan"
 [10] "bag"                "bagEarth"           "bagEarthGCV"
 [13] "bagFDA"             "bagFDAGCV"          "bam"
 [16] "bartMachine"        "bayesglm"           "binda"
 [19] "blackboost"         "blasso"             "blassoAveraged"
 [22] "bridge"             "brnn"               "BstLm"
 [25] "bstSm"              "bstTree"            "C5.0"
 [28] "C5.0Cost"           "C5.0Rules"          "C5.0Tree"
 [31] "cforest"            "chaid"              "CSimca"
 [34] "ctree"              "ctree2"             "cubist"
 [37] "dda"                "deepboost"          "DENFIS"
 [40] "dnn"                "dwdLinear"          "dwdPoly"
 [43] "dwdRadial"          "earth"              "elm"
 [46] "enet"               "evtree"             "extraTrees"
 [49] "fda"                "FH.GBML"            "FIR.DM"
 [52] "foba"               "FRBCS.CHI"          "FRBCS.W"
 [55] "FS.HGD"             "gam"                "gamboost"
 [58] "gamLoess"           "gamSpline"          "gaussprLinear"
 [61] "gaussprPoly"        "gaussprRadial"      "gbm_h2o"
 [64] "gbm"                "gcvEarth"           "GFS.FR.MOGUL"
 [67] "GFS.LT.RS"          "GFS.THRIFT"         "glm.nb"
 [70] "glm"                "glmboost"           "glmnet_h2o"
 [73] "glmnet"             "glmStepAIC"         "gpls"
 [76] "hda"                "hdda"               "hdrda"
 [79] "HYFIS"              "icr"                "J48"
 [82] "JRip"               "kernelpls"          "kknn"
 [85] "knn"                "krlsPoly"           "krlsRadial"
 [88] "lars"               "lars2"              "lasso"
 [91] "lda"                "lda2"               "leapBackward"
 [94] "leapForward"        "leapSeq"            "Linda"
 [97] "lm"                 "lmStepAIC"          "LMT"
[100] "loclda"             "logicBag"           "LogitBoost"
[103] "logreg"             "lssvmLinear"        "lssvmPoly"
```

```
[106] "lssvmRadial"          "lvq"                  "M5"
[109] "M5Rules"              "manb"                 "mda"
[112] "Mlda"                 "mlp"                  "mlpKerasDecay"
[115] "mlpKerasDecayCost"    "mlpKerasDropout"
"mlpKerasDropoutCost"
[118] "mlpML"                "mlpSGD"               "mlpWeightDecay"
[121] "mlpWeightDecayML"     "monmlp"               "msaenet"
[124] "multinom"             "mxnet"                "mxnetAdam"
[127] "naive_bayes"          "nb"                   "nbDiscrete"
[130] "nbSearch"             "neuralnet"            "nnet"
[133] "nnls"                 "nodeHarvest"          "null"
[136] "OneR"                 "ordinalNet"           "ordinalRF"
[139] "ORFlog"               "ORFpls"               "ORFridge"
[142] "ORFsvm"               "ownn"                 "pam"
[145] "parRF"                "PART"                 "partDSA"
[148] "pcaNNet"              "pcr"                  "pda"
[151] "pda2"                 "penalized"            "PenalizedLDA"
[154] "plr"                  "pls"                  "plsRglm"
[157] "polr"                 "ppr"                  "PRIM"
[160] "protoclass"           "qda"                  "QdaCov"
[163] "qrf"                  "qrnn"                 "randomGLM"
[166] "ranger"               "rbf"                  "rbfDDA"
[169] "Rborist"              "rda"                  "regLogistic"
[172] "relaxo"               "rf"                   "rFerns"
[175] "RFlda"                "rfRules"              "ridge"
[178] "rlda"                 "rlm"                  "rmda"
[181] "rocc"                 "rotationForest"       "rotationForestCp"
[184] "rpart"                "rpart1SE"             "rpart2"
[187] "rpartCost"            "rpartScore"           "rqlasso"
[190] "rqnc"                 "RRF"                  "RRFglobal"
[193] "rrlda"                "RSimca"               "rvmLinear"
[196] "rvmPoly"              "rvmRadial"            "SBC"
[199] "sda"                  "sdwd"                 "simpls"
[202] "SLAVE"                "slda"                 "smda"
[205] "snn"                  "sparseLDA"            "spikeslab"
[208] "spls"                 "stepLDA"              "stepQDA"
[211] "superpc"              "svmBoundrangeString"  "svmExpoString"
[214] "svmLinear"            "svmLinear2"           "svmLinear3"
[217] "svmLinearWeights"     "svmLinearWeights2"    "svmPoly"
[220] "svmRadial"            "svmRadialCost"        "svmRadialSigma"
[223] "svmRadialWeights"     "svmSpectrumString"    "tan"
[226] "tanSearch"            "treebag"              "vbmpRadial"
[229] "vglmAdjCat"           "vglmContRatio"        "vglmCumulative"
[232] "widekernelpls"        "WM"                   "wsrf"
[235] "xgbDART"              "xgbLinear"            "xgbTree"
[238] "xyf"
```

9 장 07 절

(1)

```
> library(rpart)
> library(randomForest)
> library(e1071)
> ucla =
read.csv('https://stats.idre.ucla.edu/stat/data/binary.csv')
> ucla$admit = factor(ucla$admit)
>
> s1 = svm(admit~., data = ucla)
> r1 = rpart(admit~., data = ucla)
> f1 = randomForest(admit~., data = ucla)
> table(predict(s1, ucla, type = 'class'), ucla$admit)

     0   1
  0 264 102
  1   9  25
> table(predict(r1, ucla, type = 'class'), ucla$admit)

     0   1
  0 249  73
  1  24  54
> table(predict(f1, ucla, type = 'class'), ucla$admit)

     0   1
  0 265  64
  1   8  63
>
> library(survival)
> clean_colon = na.omit(colon)                # 전처리: 결측값 제거
> clean_colon = clean_colon[c(TRUE, FALSE), ]  # 전처리: 홀수 번째 것만
뽑음
> clean_colon$status = factor(clean_colon$status)
>
> s2 = svm(status~rx + sex + age + obstruct + perfor + adhere +
nodes + differ + extent + surg + node4, data = clean_colon)
> r2 = rpart(status~rx + sex + age + obstruct + perfor + adhere +
nodes + differ + extent + surg + node4, data = clean_colon)
> f2 = randomForest(status~rx + sex + age + obstruct + perfor +
adhere + nodes + differ + extent + surg + node4, data = clean_colon)
>
> table(predict(s2, clean_colon, type = 'class'),
clean_colon$status)

     0   1
```

```
  0 364 178
  1  94 252
> table(predict(r2, clean_colon, type = 'class'),
clean_colon$status)

     0   1
  0 319 147
  1 139 283
> table(predict(f2, clean_colon, type = 'class'),
clean_colon$status)

     0   1
  0 441  49
  1  17 381
>
> voice = read.csv('C:/Sources/voice.csv')
>
> s3 = svm(label~., data = voice)
> r3 = rpart(label~., data = voice)
> f3 = randomForest(label~., data = voice)
>
> table(predict(s3, voice, type = 'class'), voice$label)

        female male
  female   1565   29
  male       19 1555
> table(predict(r3, voice, type = 'class'), voice$label)

        female male
  female   1551   88
  male       33 1496
> table(predict(f3, voice, type = 'class'), voice$label)

        female male
  female   1584    0
  male        0 1584
```

ucla 데이터: svm 정확률=(264+25)/400=72.25%, 결정 트리 정확률=(249+54)/400=75.75%, 랜덤포리스트 정확률=(265+63)/400=82.0%

colon 데이터: svm 정확률=(364+252)/888=69.26%, 결정 트리 정확률=(319+283)/888=67.79%, 랜덤포리스트 정확률=(441+381)/888=92.57%

voice 데이터: svm 정확률=(1565+1555)/3168=98.48%, 결정 트리 정확률

=(1551+1496)/3168=96.18%, 랜덤포리스트 정확률=(1584+1584)/3168=100%

(2)

```
> library(survival)
> clean_colon = na.omit(colon)                # 전처리: 결측값 제거
> clean_colon = clean_colon[c(TRUE, FALSE), ]  # 전처리: 홀수 번째 것만 뽑음
> clean_colon$status = factor(clean_colon$status)
>
>
form1=status~rx+sex+age+obstruct+perfor+adhere+nodes+differ+extent+surg+node4
>
form2=status~rx+sex+age+obstruct+perfor+adhere+nodes+differ+extent+surg+node4+time
>
> s1 = svm(form1,data = clean_colon)
> r1 = rpart(form1, data = clean_colon)
> f1 = randomForest(form1, data = clean_colon)
> s2 = svm(form2,data = clean_colon)
> r2 = rpart(form2, data = clean_colon)
> f3 = randomForest(form2, data = clean_colon)
> table(predict(s1, clean_colon, type = 'class'), clean_colon$status)

     0   1
  0 364 178
  1  94 252
> table(predict(s2, clean_colon, type = 'class'), clean_colon$status)

     0   1
  0 449  49
  1   9 381
> table(predict(r1, clean_colon, type = 'class'), clean_colon$status)

     0   1
  0 319 147
  1 139 283
> table(predict(r2, clean_colon, type = 'class'), clean_colon$status)

     0   1
  0 453  49
  1   5 381
```

```
> table(predict(f1, clean_colon, type = 'class'),
clean_colon$status)

     0   1
  0 438  47
  1  20 383
> table(predict(f2, clean_colon, type = 'class'),
clean_colon$status)

     0   1
  0 441  49
  1  17 381
```

svm: time 을 뺐을 때 정확률 (364+252)/888=69.26%, time 을 넣었을 때 정확률 (449+381)/888=93.47%

결정 트리: time 을 뺐을 때 정확률 (319+283)/888=67.68%, time 을 넣었을 때 정확률 (453+381)/888=93.91%

랜덤 포리스트: time 을 뺐을 때 정확률 (438+383)/888=92.45%, time 을 넣었을 때 정확률 (441+381)/888=92.57%

time 변수를 추가하면 svm, 결정 트리, 랜덤 포리스트 모두 정확률이 높아진다. 그 이유는 288 쪽의 NOTE 를 참조한다. time 을 빼고 모델링하는 것이 타당하다.

10 장 모델의 성능 평가

10 장 01 절 없음

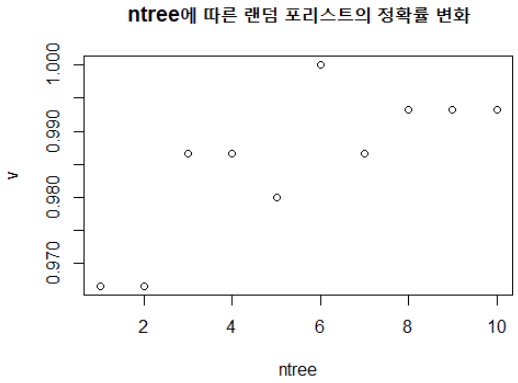
10 장 02 절

(1)

(31+24+23)/150=52.0%

(2)

```
> library(randomForest)
> p=vector(length=10)
> for (i in 1:10) {
+    f=randomForest(Species~.,data=iris,ntree=i)
+    c=table(predict(f,iris),iris$Species)
+    v[i]=sum(diag(c))/nrow(iris)
+ }
> plot(v,main='ntree 에 따른 랜덤 포리스트의 정확률 변화',xlab='ntree')
```



10 장 03 절

(1)

```
> iris1=iris[sample(nrow(iris)),] # shuffling
> n=nrow(iris1)
> n1=n*0.6
> iris_train = iris1[1:n1,]      # 앞쪽 60%
> iris_test = iris1[(n1+1):n,] # 나머지(40%)
```

(2)

height 를 설명 변수, weight 를 반응 변수로 설정한 경우 아래와 같다.

훈련집합

```
height weight
     58    115
     59    117
     60    120
     61    123
```

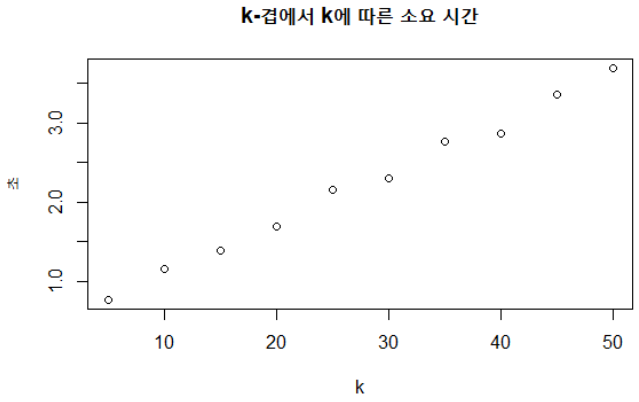
```
     62    126
     63    129
     64    132
     65    135
테스트집합
height
     66
     67
     68
     69
     70
     71
     72
     73
```

10 장 04 절

(1)

```
> library(caret)
> elapse=c()  # 공백 리스트
> x_label=c()
> for (k in seq(5,50,5)) {
+    start=Sys.time()
+    control = trainControl(method = 'cv', number = k)
+    f = train(Species~., data = iris, method = 'rf', metric =
'Accuracy', trControl = control)
+    end=Sys.time()
+    c=table(predict(f,iris),iris$Species)
+    elapse=append(elapse,as.numeric(end-start))
+    x_label=append(x_label,k)
+ }
> plot(x_label,elapse,main='k-겹에서 k 에 따른 소요 시간',xlab='k',ylab='
초')
```

위 프로그램을 실행하면 k 에 따른 소요 시간을 보여주는 다음 그래프를 얻는다. k 가 커질수록 시간이 많이 걸리는 사실을 확인할 수 있다. k 가 커지면 통계적 신뢰도는 높아진다.



(2)

```
> library(caret)
> data = iris[sample(nrow(iris)), ]   # iris 데이터의 순서를 섞는다.
> k = 5
> q = nrow(data)/k                    # k 개로 등분했을 때 부분 집합의 크기
> l = 1:nrow(data)
> accuracy = 0
> for(i in 1:k) {
+   test_list = ((i-1)*q+1):(i*q)        # i 번째를 테스트 집합으로 설정
+   testData = data[test_list, ]
+   train_list = setdiff(l, test_list)       # 나머지를 훈련 집합으로 설정
+   trainData = data[train_list, ]
+   f = train(Species~., data = trainData, method = 'rf') # 모델 학습(랜덤 포리스트)
+   p = predict(f, newdata = testData)
+   t = table(p, testData$Species)
+   current_accuracy=(t[1, 1]+t[2, 2]+t[3, 3])/length(test_list)
+   print(current_accuracy)
+   accuracy=accuracy+current_accuracy # 정확률 측정&누적
+ }
[1] 0.9333333
[1] 0.9666667
[1] 0.9333333
[1] 0.9333333
[1] 1
> (average_accuracy = accuracy/k)             # 평균 정확률
[1] 0.9533333
```

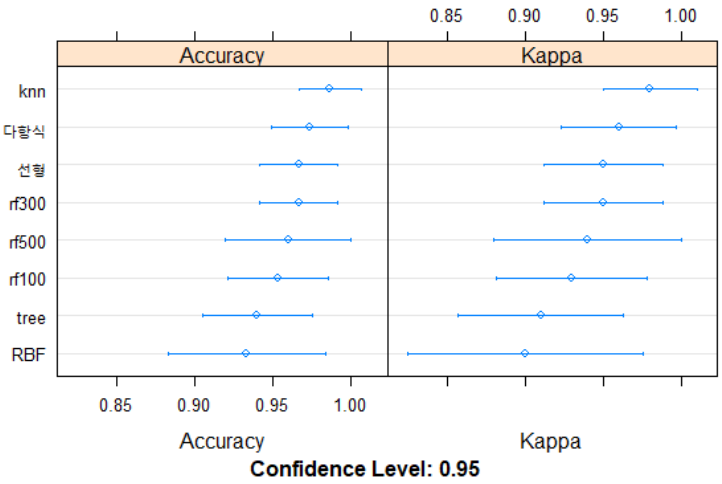
10 장 05 절

(1)

```
> library(caret)
> control = trainControl(method = 'cv', number = 10)
> formular = Species~.
```

```
> L = train(formular, data = iris, method = 'svmLinear', metric = 'Accuracy',
trControl = control)
> P = train(formular, data = iris, method = 'svmPoly', metric = 'Accuracy',
trControl =  control)
> R = train(formular, data = iris, method = 'svmRadial', metric = 'Accuracy',
trControl = control)
> f100 = train(formular, data = iris, method = 'rf', ntree = 100, metric =
'Accuracy', trControl = control)
> f300 = train(formular, data = iris, method = 'rf', ntree = 300,metric =
'Accuracy', trControl = control)
> f500 = train(formular, data = iris, method = 'rf', ntree = 500,metric =
'Accuracy', trControl = control)
> r = train(formular, data = iris, method = 'rpart', metric = 'Accuracy',
trControl = control)
> k = train(formular, data = iris, method = 'knn', metric = 'Accuracy', trControl
= control)
> resamp = resamples(list(선형 = L, 다항식 = P, RBF = R, rf100 = f100, rf300 = f300,
rf500 = f500, tree = r, knn = k))
> summary(resamp)

Call:
summary.resamples(object = resamp)

Models: 선형, 다항식, RBF, rf100, rf300, rf500, tree, knn
Number of resamples: 10

Accuracy
          Min.   1st Qu.    Median      Mean   3rd Qu. Max. NA's
선형   0.9333333 0.9333333 0.9666667 0.9666667 1.0000000    1    0
다항식 0.9333333 0.9333333 1.0000000 0.9733333 1.0000000    1    0
RBF    0.8000000 0.8833333 0.9333333 0.9333333 1.0000000    1    0
rf100  0.8666667 0.9333333 0.9333333 0.9533333 1.0000000    1    0
rf300  0.9333333 0.9333333 0.9666667 0.9666667 1.0000000    1    0
rf500  0.8666667 0.9333333 1.0000000 0.9600000 1.0000000    1    0
tree   0.8666667 0.9333333 0.9333333 0.9400000 0.9833333    1    0
knn    0.9333333 1.0000000 1.0000000 0.9866667 1.0000000    1    0

Kappa
      Min. 1st Qu. Median Mean 3rd Qu. Max. NA's
선형    0.9   0.900   0.95 0.95   1.000    1    0
다항식  0.9   0.900   1.00 0.96   1.000    1    0
RBF     0.7   0.825   0.90 0.90   1.000    1    0
rf100   0.8   0.900   0.90 0.93   1.000    1    0
rf300   0.9   0.900   0.95 0.95   1.000    1    0
rf500   0.8   0.900   1.00 0.94   1.000    1    0
tree    0.8   0.900   0.90 0.91   0.975    1    0
knn     0.9   1.000   1.00 0.98   1.000    1    0

>
> sort(resamp, decreasing = TRUE)
[1] "knn"    "다항식" "선형"   "rf300"  "rf500"  "rf100"  "tree"   "RBF"
```

```
>
> dotplot(resamp)
```



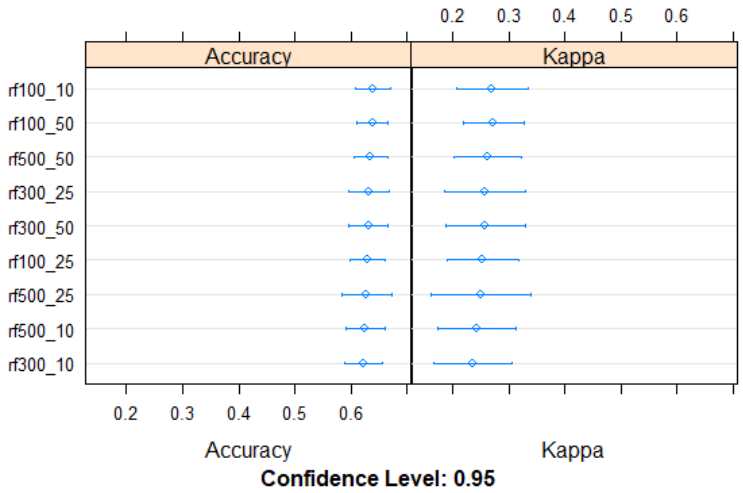
(2)

```
> library(survival)
> clean_colon = na.omit(colon)
> clean_colon = clean_colon[c(TRUE, FALSE), ]
> clean_colon$status = factor(clean_colon$status)
>
> control = trainControl(method = 'cv', number = 10)
> formular =
status~rx+sex+age+obstruct+perfor+adhere+nodes+differ+extent+surg+node4
>
> f100_10 = train(formular, data = clean_colon, method = 'rf', ntree =
100,maxnodes=10, metric = 'Accuracy', trControl = control)
> f100_25 = train(formular, data = clean_colon, method = 'rf', ntree =
100,maxnodes=25, metric = 'Accuracy', trControl = control)
> f100_50 = train(formular, data = clean_colon, method = 'rf', ntree =
100,maxnodes=50, metric = 'Accuracy', trControl = control)
> f300_10 = train(formular, data = clean_colon, method = 'rf', ntree =
300,maxnodes=10, metric = 'Accuracy', trControl = control)
> f300_25 = train(formular, data = clean_colon, method = 'rf', ntree =
300,maxnodes=25, metric = 'Accuracy', trControl = control)
> f300_50 = train(formular, data = clean_colon, method = 'rf', ntree =
300,maxnodes=50, metric = 'Accuracy', trControl = control)
> f500_10 = train(formular, data = clean_colon, method = 'rf', ntree =
500,maxnodes=10, metric = 'Accuracy', trControl = control)
> f500_25 = train(formular, data = clean_colon, method = 'rf', ntree =
500,maxnodes=25, metric = 'Accuracy', trControl = control)
> f500_50 = train(formular, data = clean_colon, method = 'rf', ntree =
500,maxnodes=50, metric = 'Accuracy', trControl = control)
>
> resamp =
resamples(list(rf100_10=f100_10,rf100_25=f100_25,rf100_50=f100_50,rf300_10=f300_10
,rf300_25=f300_25,rf300_50=f300_50,rf500_10=f500_10,rf500_25=f500_25,rf500_50=f500
_50))
> summary(resamp)
```

Call:

```
summary.resamples(object = resamp)

Models: rf100_10, rf100_25, rf100_50, rf300_10, rf300_25, rf300_50,
rf500_10, rf500_25, rf500_50
Number of resamples: 10

Accuracy
            Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
rf100_10 0.5617978 0.6175306 0.6348315 0.6385470 0.6488764 0.7191011
0
rf100_25 0.5505618 0.6197331 0.6292135 0.6295072 0.6345761 0.7303371
0
rf100_50 0.5730337 0.6225741 0.6460674 0.6385342 0.6572395 0.6853933
0
rf300_10 0.5505618 0.5814607 0.6271067 0.6216675 0.6488764 0.7078652
0
rf300_25 0.5617978 0.5983146 0.6404494 0.6318309 0.6675498 0.7078652
0
rf300_50 0.5340909 0.6123596 0.6348315 0.6315884 0.6601124 0.7078652
0
rf500_10 0.5393258 0.6136364 0.6235955 0.6260981 0.6376404 0.7303371
0
rf500_25 0.5393258 0.5834397 0.6348315 0.6283453 0.6572395 0.7415730
0
rf500_50 0.5795455 0.6095506 0.6292135 0.6351124 0.6629533 0.6966292
0

Kappa
              Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
rf100_10 0.11925907 0.2276117 0.2612884 0.2699384 0.2919842
0.4310918    0
rf100_25 0.09598781 0.2318804 0.2501915 0.2530283 0.2631119
0.4584178    0
rf100_50 0.13724490 0.2403349 0.2851301 0.2725628 0.3133550
0.3662258    0
rf300_10 0.08483290 0.1573063 0.2455644 0.2357303 0.2909011
0.4105960    0
rf300_25 0.11250320 0.1919072 0.2767751 0.2570724 0.3289597
0.4087890    0
rf300_50 0.06139438 0.2194539 0.2646144 0.2581456 0.3120360
0.4150657    0
rf500_10 0.06555698 0.2179822 0.2379193 0.2424409 0.2657180
0.4508997    0
rf500_25 0.07126495 0.1577176 0.2624038 0.2501350 0.3104260
0.4774062    0
rf500_50 0.15472482 0.2147979 0.2484607 0.2625389 0.3190477
0.3874586    0

>
> sort(resamp, decreasing = TRUE)
```

```
[1] "rf100_10" "rf100_50" "rf500_50" "rf300_25" "rf300_50"
"rf100_25"
[7] "rf500_25" "rf500_10" "rf300_10"
>
> dotplot(resamp)
```



이번 실험 결과에서는 rf100_10, 즉 ntree=100, maxnodes=10 으로 설정한 랜덤 포리스트가 평균 정확률 63.8547%로서 가장 높은 성능을 보였다.

(3)

랜덤 포리스트는 난수를 사용하므로 실행할 때마다 조금씩 다른 결과를 생성한다. 실행할 때마다 set_seed(1)을 먼저 실행하면 항상 같은 결과를 얻는다. 이때 1 은 다른 숫자를 써도 되는데, 실행할 때마다 같은 숫자를 사용해야 한다.

10 장 06 절

(1)

O O X O

(2)

가방을 열어 검사하는 경우

(3)

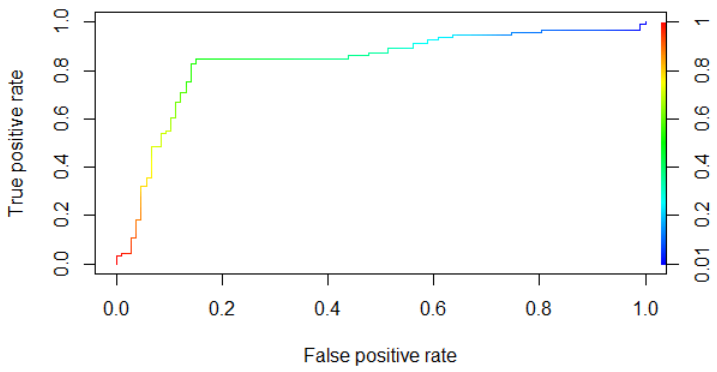
TP=2, FN: 2, FP: 3, TN: 5

정밀도=2/(2+3)=0.4, 재현률=2/(2+2)=0.5, 특이도=3/(3+5)=0.375, 민감도=2/(2+2)=0.5

10 장 07 절

(1)

```
> library(ROCR)
> data(ROCR.simple)
> p=prediction(ROCR.simple$predictions, ROCR.simple$labels)
> roc=performance(p,"tpr","fpr")
> plot(roc,colorize=TRUE)
> auc=performance(p,measure='auc')
> auc@y.values
[[1]]
[1] 0.8341875
```

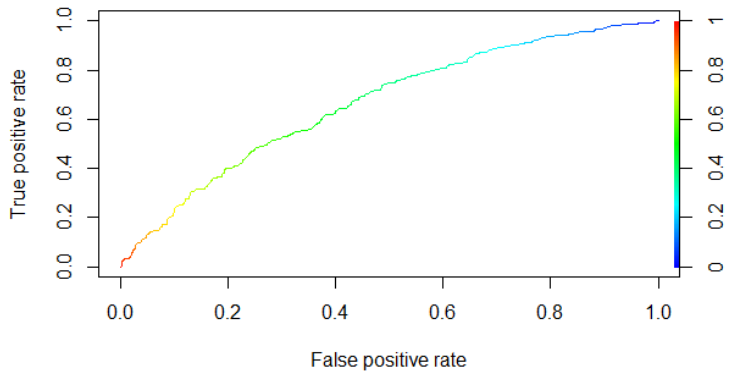


AUC 는 0.8341875 이다.

(2)

```
> library(survival)
> library(randomForest)
> clean_colon = na.omit(colon)
> clean_colon = clean_colon[c(TRUE, FALSE), ]
> clean_colon$status = factor(clean_colon$status)
>
```

```
> formular =
status~rx+sex+age+obstruct+perfor+adhere+nodes+differ+extent+surg+no
de4
> f100 = randomForest(formular, data = clean_colon, ntree = 100)
> pred=predict(f100,data=clean_colon,type='prob')
> p=prediction(pred[,2], clean_colon$status) # pred[,2]는 환자 확률
> roc=performance(p,"tpr","fpr")
> plot(roc,colorize=TRUE)
> auc=performance(p,measure='auc')
> auc@y.values
[[1]]
[1] 0.663692
```



AUC 는 0.663692 이다.

11 장 텍스트 마이닝

11 장 01 절

(1)

In the field of computer science, artificial intelligence (AI), sometimes called machine intelligence, is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans and other animals.

모두 소문자로 변경 →

in the field of computer science, artificial intelligence (ai), sometimes called machine intelligence, is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans and other animals.

숫자 제거 →

in the field of computer science, artificial intelligence (ai), sometimes called machine intelligence, is intelligence demonstrated by machines, in contrast to the natural intelligence displayed by humans and other animals.

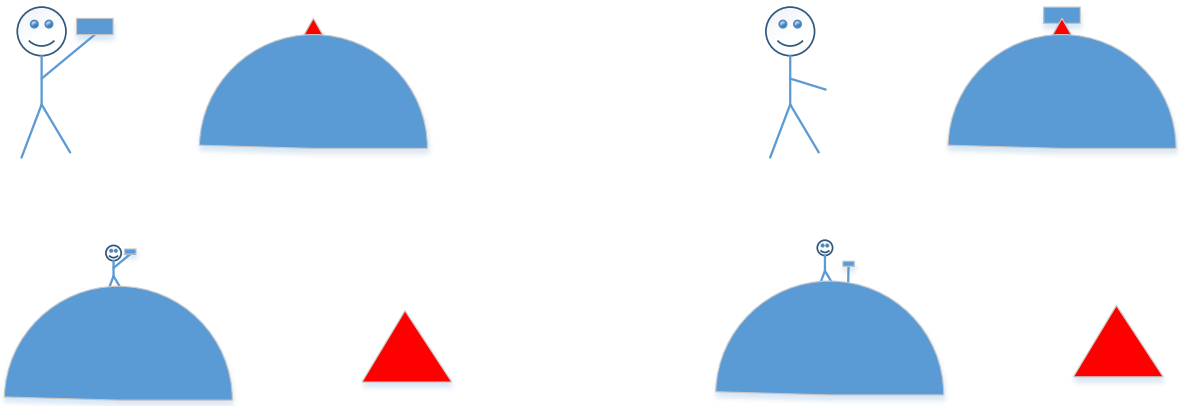
불용어 제거 →

field computer science, artificial intelligence (ai), sometimes called machine intelligence, intelligence demonstrated machines, contrast natural intelligence displayed humans other animals.
구두점 제거 →
field computer science artificial intelligence (ai) sometimes called machine intelligence intelligence demonstrated machines contrast natural intelligence displayed humans other animals
어간 추출 →
field computer science artificial intelligence (ai) sometimes call machine intelligence intelligence demonstrate machine contrast natural intelligence display human other animal

(2)

인공지능(人工知能, 영어: artificial intelligence, AI)은 기계로부터 만들어진 지능을 말한다.[1] 컴퓨터 공학에서 이상적인 지능을 갖춘 존재, 혹은 시스템에 의해 만들어진 지능, 즉 인공적인 지능을 뜻한다.
모두 소문자로 변경 →
인공지능(人工知能, 영어: artificial intelligence, ai)은 기계로부터 만들어진 지능을 말한다.[1] 컴퓨터 공학에서 이상적인 지능을 갖춘 존재, 혹은 시스템에 의해 만들어진 지능, 즉 인공적인 지능을 뜻한다.
숫자 제거 →
인공지능(人工知能, 영어: artificial intelligence, ai)은 기계로부터 만들어진 지능을 말한다. 컴퓨터 공학에서 이상적인 지능을 갖춘 존재, 혹은 시스템에 의해 만들어진 지능, 즉 인공적인 지능을 뜻한다.
불용어 제거 →
인공지능(人工知能, 영어: artificial intelligence, ai)은 기계로부터 만들어진 지능을 말한다. 컴퓨터 공학에서 이상적인 지능을 갖춘 존재, 시스템에 만들어진 지능, 인공적인 지능을 뜻한다.
구두점 제거 →
인공지능(人工知能 영어 artificial intelligence ai)은 기계로부터 만들어진 지능을 말한다 컴퓨터 공학에서 이상적인 지능을 갖춘 존재 시스템에 만들어진 지능 인공적인 지능을 뜻한다
어간 추출 →
인공지능(人工知能 영어 artificial intelligence ai) 기계 만들다 지능 말하다 컴퓨터 공학 이상적 지능 갖추다 존재 시스템 만들다 지능 인공적 지능 뜻하다

(3)



(4)

아버지가 방에 들어가신다.

아버지 가방에 들어가신다. (아버지가 가방에 들어가신다.)

아버지 가방에 들어가신다. (아버지의 가방에 들어가신다.)

11 장 02 절

(1) 생략

(2)

```
> library(RCurl)
> library(XML)
> library(tm)
> library(SnowballC)
> t = readLines('https://en.wikipedia.org/wiki/Data_science')
> d = htmlParse(t, asText = TRUE)
> clean_doc = xpathSApply(d,"//p", xmlValue)
>
> doc = Corpus(VectorSource(clean_doc))
> doc = tm_map(doc, content_transformer(tolower))
Warning message:
In tm_map.SimpleCorpus(doc, content_transformer(tolower)) :
  transformation drops documents
> doc = tm_map(doc, removeNumbers)
Warning message:
In tm_map.SimpleCorpus(doc, removeNumbers) : transformation drops documents
> doc = tm_map(doc, removeWords, stopwords('english'))
Warning message:
In tm_map.SimpleCorpus(doc, removeWords, stopwords("english")) :
  transformation drops documents
> doc = tm_map(doc, removePunctuation)
Warning message:
In tm_map.SimpleCorpus(doc, removePunctuation) :
  transformation drops documents
> doc = tm_map(doc, stripWhitespace)
Warning message:
In tm_map.SimpleCorpus(doc, stripWhitespace) :
  transformation drops documents
>
> dtm1 = DocumentTermMatrix(doc) # 기본값: control=list(weighting=weightTf)
> dtm2 = DocumentTermMatrix(doc, control=list(weighting=weightTfIdf))
Warning message:
In weighting(x) : empty document(s): 1
> inspect(dtm1)
<<DocumentTermMatrix (documents: 16, terms: 582)>>
Non-/sparse entries: 795/8517
Sparsity           : 91%
Maximal term length: 22
Weighting          : term frequency (tf)
Sample             :
    Terms
Docs big data field many now science scientists statistical statistics term
```

```
  12   1   14     0    0   0       9          0          4          0    0
  13   4   13     0    1   0       4          3          2          1    4
  14   0    9     1    1   2      10          0          0          0    0
  15   2   13     3    2   1       7          1          0          3    0
  16   1    8     0    0   0       6          0          1          0    0
  3    0    5     0    1   1       6          0          0          2    0
  4    0    5     0    3   4       5          0          0          3    3
  7    0    7     0    0   0       5          1          2          3    1
  8    0    6     3    0   0       3          0          1          2    0
  9    0    8     0    0   0       5          2          1          0    0
> inspect(dtm2)
<<DocumentTermMatrix (documents: 16, terms: 582)>>
Non-/sparse entries: 795/8517
Sparsity           : 91%
Maximal term length: 22
Weighting          : term frequency - inverse document frequency (normalized) (tf-
idf)
Sample             :
    Terms
Docs algorithms classification conference     field    methods naur online
statistics    systems       term
  11          0     0.0000000  0.0000000 0.00000000 0.00000000    0      0 0.00000000
0.00000000 0.03930660
  13          0     0.0000000  0.0000000 0.00000000 0.00000000    0      0 0.01141159
0.00000000 0.04564637
  14          0     0.0000000  0.0000000 0.01869159 0.00000000    0      0 0.00000000
0.00000000 0.00000000
  15          0     0.0000000  0.0000000 0.04878049 0.00000000    0      0 0.03451311
0.00000000 0.00000000
  16          0     0.0000000  0.0000000 0.00000000 0.00000000    0      0 0.00000000
0.00000000 0.00000000
  4           0     0.0000000  0.0000000 0.00000000 0.00000000    0      0 0.04376405
0.00000000 0.04376405
  6           0     0.2758621  0.2068966 0.00000000 0.04879440    0      0 0.00000000
0.00000000 0.09758879
  7           0     0.0000000  0.0000000 0.00000000 0.00000000    0      0 0.06431989
0.00000000 0.02143996
  8           0     0.0000000  0.0000000 0.10714286 0.02526853    0      0 0.05053705
0.00000000 0.00000000
  9           0     0.0000000  0.0000000 0.00000000 0.01521546    0      0 0.00000000
0.03225806 0.00000000
```

Tf (term frequency)는 특정 문서에서 구한 단어의 발생 빈도를 나타낸다. Df (document frequency)는 모든 문서의 집합, 즉 말뭉치에서 구한 단어의 발생 빈도이다. Df 가 큰 단어는 중요도가 낮다고 보고, Tf 에 Df 의 역수, 즉 Idf (inverse document frequency)를 곱한 것이 Tfidf 이다. 보통 정보검색에서는 Tf 보다 Tfidf 가 더 정확하다고 평가하고 더 많이 사용한다.

위의 실험에서 Tf 를 사용한 경우에는 big, data 라는 단어가 상위에 위치한 반면에 Tfidf 를 사용한 경우에는 algorithms, classification 이 상위를 차지하였다.

(3)

```
> library(tm)
> library(SnowballC)
> t=c('Data science is exciting and motivating.','I like literature class and science class.','What is
data science?')
>
> doc = Corpus(VectorSource(t))
> doc = tm_map(doc, content_transformer(tolower))
Warning message:
In tm_map.SimpleCorpus(doc, content_transformer(tolower)) :
  transformation drops documents
> doc = tm_map(doc, removeNumbers)
Warning message:
In tm_map.SimpleCorpus(doc, removeNumbers) : transformation drops documents
> doc = tm_map(doc, removeWords, stopwords('english'))
Warning message:
In tm_map.SimpleCorpus(doc, removeWords, stopwords("english")) :
  transformation drops documents
> doc = tm_map(doc, removePunctuation)
Warning message:
In tm_map.SimpleCorpus(doc, removePunctuation) :
  transformation drops documents
> doc = tm_map(doc, stripWhitespace)
Warning message:
In tm_map.SimpleCorpus(doc, stripWhitespace) :
  transformation drops documents
>
> dtm = DocumentTermMatrix(doc)
> dim(dtm)
[1] 3 7
> inspect(dtm)
<<DocumentTermMatrix (documents: 3, terms: 7)>>
Non-/sparse entries: 10/11
Sparsity           : 52%
Maximal term length: 10
Weighting          : term frequency (tf)
Sample             :
    Terms
Docs class data exciting like literature motivating science
   1     0    1       1    0          0          1       1
   2     2    0       0    1          1          0       1
   3     0    1       0    0          0          0       1
```

(힌트: 불용어를 제거하는 `> doc = tm_map(doc, removeWords, stopwords('english'))` 명령어를 제거하면 `I` 와 `What` 도 남게 되어 책과 같다.)

11 장 03 절

(1)

'circle' (기본값), 'cardioid', 'diamond', 'triangle-forward', 'triangle', 'pentagon', 'star'

(2)

```
> library(RCurl)
> library(XML)
> library(tm)
> library(SnowballC)
> t = readLines('https://en.wikipedia.org/wiki/Data_science')
> d = htmlParse(t, asText = TRUE)
> clean_doc = xpathSApply(d,"//p", xmlValue)
>
> doc = Corpus(VectorSource(clean_doc))
> doc = tm_map(doc, content_transformer(tolower))
Warning message:
In tm_map.SimpleCorpus(doc, content_transformer(tolower)) :
  transformation drops documents
> doc = tm_map(doc, removeNumbers)
Warning message:
In tm_map.SimpleCorpus(doc, removeNumbers) : transformation drops documents
> doc = tm_map(doc, removeWords, stopwords('english'))
Warning message:
In tm_map.SimpleCorpus(doc, removeWords, stopwords("english")) :
  transformation drops documents
> doc = tm_map(doc, removePunctuation)
Warning message:
In tm_map.SimpleCorpus(doc, removePunctuation) :
  transformation drops documents
> doc = tm_map(doc, stripWhitespace)
Warning message:
In tm_map.SimpleCorpus(doc, stripWhitespace) :
  transformation drops documents
>
> dtm = DocumentTermMatrix(doc)
> m = as.matrix(dtm)
> v = sort(colSums(m), decreasing = TRUE)
> d = data.frame(word = names(v), freq = v)
> d1 = d[1:200, ]              # 500 개 단어만 표시
> wordcloud2(d1,minRotation=pi/4,maxRotation=3*pi/4,rotateRatio=0.2)
```

`minRotation=pi/4,maxRotation=3*pi/4,rotateRatio=0.2` 로 설정하여, 단어가 기운 각도는 45~135 도로 유지하고, 20%를 기울게 하였다. 단어가 200 개이니 대략 40

개의 단어가 45~135 도로 기울어져 있다.

(3)

Oceania 의 비중이 너무 작아 눈으로 확인할 수 없을 정도로 작은 폰트로 그려진 탓이다.

(4) 생략

11 장 04 절

(1)

```
> library(text2vec)
> library(caret)
> library(tm)
> library(SnowballC)
>
> train_list = createDataPartition(y= movie_review$sentiment, p = 0.6, list =
FALSE)
> mtrain = movie_review[train_list, ]
> mtest = movie_review[-train_list, ]
>
> doc = Corpus(VectorSource(mtrain$review))
> doc = tm_map(doc, content_transformer(tolower))
Warning message:
In tm_map.SimpleCorpus(doc, content_transformer(tolower)) :
  transformation drops documents
> doc = tm_map(doc, removeNumbers)
Warning message:
In tm_map.SimpleCorpus(doc, removeNumbers) : transformation drops documents
> doc = tm_map(doc, removeWords, stopwords('english'))
Warning message:
In tm_map.SimpleCorpus(doc, removeWords, stopwords("english")) :
  transformation drops documents
> doc = tm_map(doc, removePunctuation)
Warning message:
In tm_map.SimpleCorpus(doc, removePunctuation) :
  transformation drops documents
> doc = tm_map(doc, stripWhitespace)
Warning message:
In tm_map.SimpleCorpus(doc, stripWhitespace) :
  transformation drops documents
> dtm = DocumentTermMatrix(doc)
>
> dtm_small = removeSparseTerms(dtm, 0.90)
> X = as.matrix(dtm_small)
> dataTrain = as.data.frame(cbind(mtrain$sentiment, X))
> dataTrain$V1 = as.factor(dataTrain$V1)
> colnames(dataTrain)[1] = 'y'
>
```

```
> library(rpart)
> library(randomForest)
> library(e1071)
> r = rpart(y~., data = dataTrain)
> f = randomForest(y~., data = dataTrain)
> s = svm(y~., data = dataTrain)
>
> # 테스트 집합으로 DTM 구축
> docTest = Corpus(VectorSource(mtest$review))
> docTest = tm_map(docTest, content_transformer(tolower))
Warning message:
In tm_map.SimpleCorpus(docTest, content_transformer(tolower)) :
  transformation drops documents
> docTest = tm_map(docTest, removeNumbers)
Warning message:
In tm_map.SimpleCorpus(docTest, removeNumbers) :
  transformation drops documents
> docTest = tm_map(docTest, removeWords, stopwords('english'))
Warning message:
In tm_map.SimpleCorpus(docTest, removeWords, stopwords("english")) :
  transformation drops documents
> docTest = tm_map(docTest, removePunctuation)
Warning message:
In tm_map.SimpleCorpus(docTest, removePunctuation) :
  transformation drops documents
> docTest = tm_map(docTest, stripWhitespace)
Warning message:
In tm_map.SimpleCorpus(docTest, stripWhitespace) :
  transformation drops documents
>
> dtmTest = DocumentTermMatrix(docTest,
control=list(dictionary=dtm_small$dimnames$Terms))
>
> X = as.matrix(dtmTest)
> dataTest = as.data.frame(cbind(mtest$sentiment, X))
> dataTest$V1 = as.factor(dataTest$V1)
> colnames(dataTest)[1] = 'y'
>
> pr = predict(r, newdata = dataTest, type = 'class')
> table(pr, dataTest$y)

pr   0   1
  0 550 262
  1 428 760
> pf = predict(f, newdata = dataTest)
> table(pf, dataTest$y)

pf   0   1
  0 691 271
  1 287 751
> ps = predict(s, newdata = dataTest)
> table(ps, dataTest$y)

ps   0   1
  0 664 255
  1 314 767
```

결정 트리 정확률=(550+760)/2000=65.5%

랜덤 포리스트 정확률=(691+751)/2000=72.1%

Svm 정확률=(664+767)/2000=71.55%

랜덤 포리스트가 가장 높은 성능을 보인다.

(2) 생략

11 장 05 절

(1)

훨씬 촘촘하게 단어가 배치되어 있다. str 명령어로 확인해보면, 500 개를 추출하는 명령어를 생략하면 1443 개 단어가 사용되어 약 3 배의 단어가 단어 구름에 배치된다.

(2)

```
> library(tm)
> library(XML)
> library(wordcloud2)
> library(SnowballC)
> library(RCurl)
> t =
readLines('https://ko.wikipedia.org/wiki/%EB%B9%85_%EB%8D%B0%EC%9D%B4%ED%84%B0')
> d = htmlParse(t, asText = TRUE)
```

```
> clean_doc = xpathSApply(d, "//p", xmlValue)
>
> doc = Corpus(VectorSource(clean_doc))
> doc = tm_map(doc, content_transformer(tolower))
Warning message:
In tm_map.SimpleCorpus(doc, content_transformer(tolower)) :
  transformation drops documents
> doc = tm_map(doc, removeNumbers)
Warning message:
In tm_map.SimpleCorpus(doc, removeNumbers) : transformation drops documents
> doc = tm_map(doc, removePunctuation)
Warning message:
In tm_map.SimpleCorpus(doc, removePunctuation) :
  transformation drops documents
> doc = tm_map(doc, stripWhitespace)
Warning message:
In tm_map.SimpleCorpus(doc, stripWhitespace) :
  transformation drops documents
>
> dtm = DocumentTermMatrix(doc)
> m = as.matrix(dtm)
> v = sort(colSums(m), decreasing = TRUE)
> d = data.frame(word = names(v), freq = v)
> d1 = d[1:50, ]              # 500개 단어만 표시
> wordcloud2(d1)
```

11 장 06 절

(1)

```
> library(KoNLP)
>
> useSejongDic()
Backup was just finished!
370957 words dictionary was built.
```

```
> s='너에게 묻는다 연탄재 함부로 발로 차지 마라 너는 누구에게 한번이라도 뜨거운 사람이었느냐'
> SimplePos09(s)
$너에게
[1] "너/N+에게/J"

$묻는다
[1] "묻/P+는다/E"

$연탄재
[1] "연탄재/N"

$함부로
[1] "함부로/M"

$발로
[1] "발/N+로/J"

$차지
[1] "차/N+이/J+지/E"

$마라
[1] "마르/P+아/E"

$너는
[1] "너/N+는/J"

$누구에게
[1] "누구/N+에게/J"

$한번이라도
[1] "한번/N+이라도/J"

$뜨거운
[1] "뜨겁/P+은/E"

$사람이었느
[1] "사람이었느/N"

$냐
[1] "냐/N"
```

```
> SimplePos22(s)
$너에게
[1] "너/NP+에게/JC"

$묻는다
[1] "묻/PV+는다/EF"

$연탄재
[1] "연탄재/NC"

$함부로
[1] "함부로/MA"

$발로
[1] "발/NC+로/JC"

$차지
[1] "차/NC+이/JP+지/EC"

$마라
[1] "마르/PV+아/EC"

$너는
[1] "너/NP+는/JX"

$누구에게
[1] "누구/NP+에게/JC"

$한번이라도
[1] "한/NN+번/NB+이라도/JX"

$뜨거운
[1] "뜨겁/PA+은/ET"

$사람이었느
[1] "사람이었느/NC"

$냐
[1] "냐/NC"

> MorphAnalyzer(s)
```

```
$너에게
[1] "너/npp+에게/jca"           "너/npp+에/jca+이/jp+게/ecc"
[3] "너/npp+에/jca+이/jp+게/ecs" "너/npp+에/jca+이/jp+게/ecx"
[5] "너/npp+에/jca+이/jp+게/ef"

$묻는다
[1] "묻/pvg+는다/ef"

$연탄재
[1] "연탄재/ncn"     "연탄/ncn+재/ncn"

$함부로
[1] "함부로/mag"

$발로
[1] "발로/ncpa"    "발/nbu+로/jca" "발/ncn+로/jca"

$차지
 [1] "차지/ncpa"          "차/nbn+이/jp+지/ecs" "차/nbn+이/jp+지/ecx"
 [4] "차/nbn+이/jp+지/ef"  "차/nbu+이/jp+지/ecs" "차/nbu+이/jp+지/ecx"
 [7] "차/nbu+이/jp+지/ef"  "차/ncn+이/jp+지/ecs" "차/ncn+이/jp+지/ecx"
[10] "차/ncn+이/jp+지/ef"  "차/npd+이/jp+지/ecs" "차/npd+이/jp+지/ecx"
[13] "차/npd+이/jp+지/ef"  "차/paa+지/ecs"      "차/paa+지/ecx"
[16] "차/paa+지/ef"        "차/pvg+지/ecs"      "차/pvg+지/ecx"
[19] "차/pvg+지/ef"

$마라
[1] "마/nbu+이/jp+라/ecs" "마/nbu+이/jp+라/ef"  "마/ncn+이/jp+라/ecs"
[4] "마/ncn+이/jp+라/ef"  "마르/pvg+아/ecs"    "마르/pvg+아/ecx"
[7] "마르/pvg+아/ef"

$너는
[1] "너/npp+는/jxc" "널/pvg+는/etm"

$누구에게
[1] "누구/npp+에게/jca"           "누구/npp+에/jca+이/jp+게/ecc"
[3] "누구/npp+에/jca+이/jp+게/ecs" "누구/npp+에/jca+이/jp+게/ecx"
[5] "누구/npp+에/jca+이/jp+게/ef"

$한번이라도
```

```
[1] "한번/mag+이라도/jxc"
[2] "한/nnc+번/nbu+이라도/jxc"
[3] "한/nnc+번/nbu+이라/jca+도/jxc"
[4] "한/nnc+번/nbu+이/jcc+라도/jxc"
[5] "한/nnc+번/nbu+이/jcs+라도/jxc"
[6] "한/nnc+번/nbu+이/jp+라도/ecs"
[7] "한/nnc+번/nbu+이/jp+라/ecs+도/jxc"

$뜨거운
[1] "뜨겁/paa+은/etm"

$사람이었느
[1] "사람이었느/ncn" "사람이었느/nqq"

$냐
[1] "냐/ncn" "냐/nqq"
```

(2)

```
> library(KoNLP)
>
> s='너에게 묻는다 연탄재 함부로 발로 차지 마라 너는 누구에게 한번이라도 뜨거운 사
람이었느냐'
> useSystemDic()
Backup was just finished!
283949 words dictionary was built.
> SimplePos09(s)
$너에게
[1] "너/N+에게/J"

$묻는다
[1] "묻/P+는다/E"

$연탄재
[1] "연탄재/N"

$함부로
[1] "함부로/M"

$발로
[1] "발/N+로/J"
```

```
$차지
[1] "차/N+이/J+지/E"

$마라
[1] "마르/P+아/E"

$너는
[1] "너/N+는/J"

$누구에게
[1] "누구/N+에게/J"

$한번이라도
[1] "한번/N+이라도/J"

$뜨거운
[1] "뜨겁/P+은/E"

$사람이었느
[1] "사람이었느/N"

$냐
[1] "냐/N"

> useNIADic()
Backup was just finished!
983012 words dictionary was built.
> SimplePos09(s)
$너에게
[1] "너/N+에게/J"

$묻는다
[1] "묻/P+는다/E"

$연탄재
[1] "연탄재/N"

$함부로
[1] "함부로/M"

$발로
[1] "발/N+로/J"
```

```
$차지
[1] "차/N+이/J+지/E"

$마라
[1] "마르/P+아/E"

$너는
[1] "너/N+는/J"

$누구에게
[1] "누구/N+에게/J"

$한번이라도
[1] "한번/N+이라도/J"

$뜨거운
[1] "뜨겁/P+은/E"

$사람이었느
[1] "사람이었느/N"

$냐
[1] "냐/N"
```

이 문장에 관한한 두 사전은 같은 결과를 출력한다.

12 장 구글 플레이 스토어 앱을 이용한 실전 프로젝트

12 장 01 절 없음

12 장 02 절

(1)

원본 데이터를 View 명령으로 확인해보면,

	App	Category	Rating	Reviews	Size	Installs	Type	Price	Content.Rating
10468	FI CFL	FINANCE	3.7	112	3.9M	10,000+	Free	0	Everyone
10469	Tassa.fi Finland	LIFESTYLE	3.6	346	7.5M	50,000+	Free	0	Everyone
10470	TownWiFi | Wi-Fi Everywhere	COMMUNICATION	3.9	2372	58M	500,000+	Free	0	Everyone
10471	Jazz Wi-Fi	COMMUNICATION	3.4	49	4.0M	10,000+	Free	0	Everyone
10472	Xposed Wi-Fi-Pwd	PERSONALIZATION	3.5	1042	404k	100,000+	Free	0	Everyone
10473	Life Made WI-Fi Touchscreen Photo Frame	1.9	19.0	3.0M	1,000+	Free	0	Everyone	
10474	osmino Wi-Fi: free WiFi	TOOLS	4.2	134203	4.1M	10,000,000+	Free	0	Everyone
10475	Sat-Fi Voice	COMMUNICATION	3.4	37	14M	1,000+	Free	0	Everyone
10476	Wi-Fi Visualizer	TOOLS	3.9	132	2.6M	50,000+	Free	0	Everyone
10477	Lennox iComfort Wi-Fi	LIFESTYLE	3.0	552	7.6M	50,000+	Free	0	Everyone
10478	Sci-Fi Sounds and Ringtones	PERSONALIZATION	3.6	128	11M	10,000+	Free	0	Everyone

그림과 같이 10,473 행의 데이터는 Category 데이터가 없고, 그 이후의 속성(Rating, Reviews, Size 등)이 한 칸씩 당겨져 채워져 있는 것을 볼 수 있다. 이대로 데이터를 처리하게 되면 다른 속성들의 데이터형의 일관성에 문제가 발생한다.

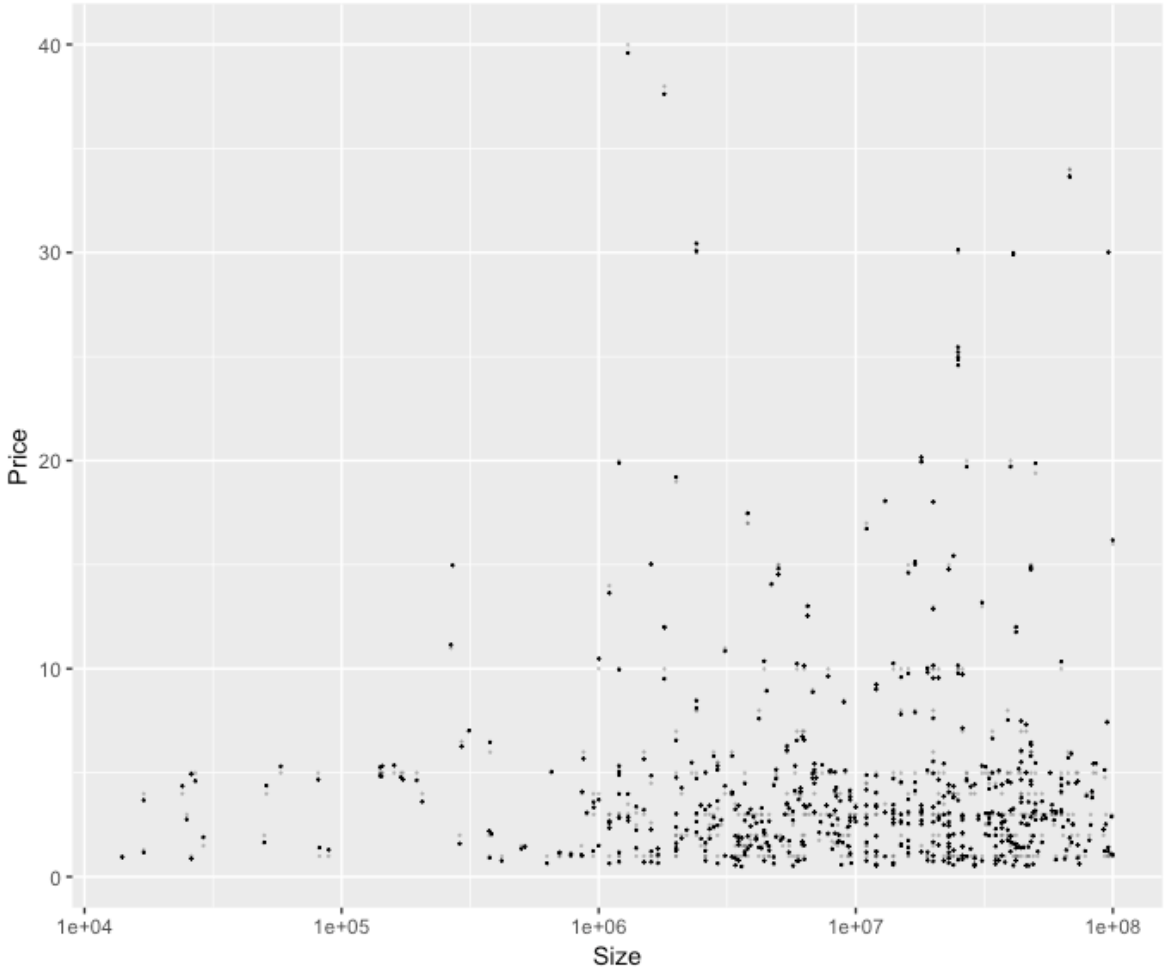
(2)

숫자형 데이터로 가정하고 그대로 처리하면 R은 이를 제대로 인식하지 못할 뿐만 아니라, Mega byte 에 해당하는 10의 6제곱 등을 처리하지 않은 상태에서는 정확한 수치로 처리할 수도 없게 된다.

12 장 03 절

(1)

```
> x %>% filter(Type=="Paid" & Price<60) %>% ggplot(aes(Size, Price)) +
geom_point(alpha=0.2, size=0.1) + geom_jitter(size=0.1, height = 0.5) + scale_x_log10()
```



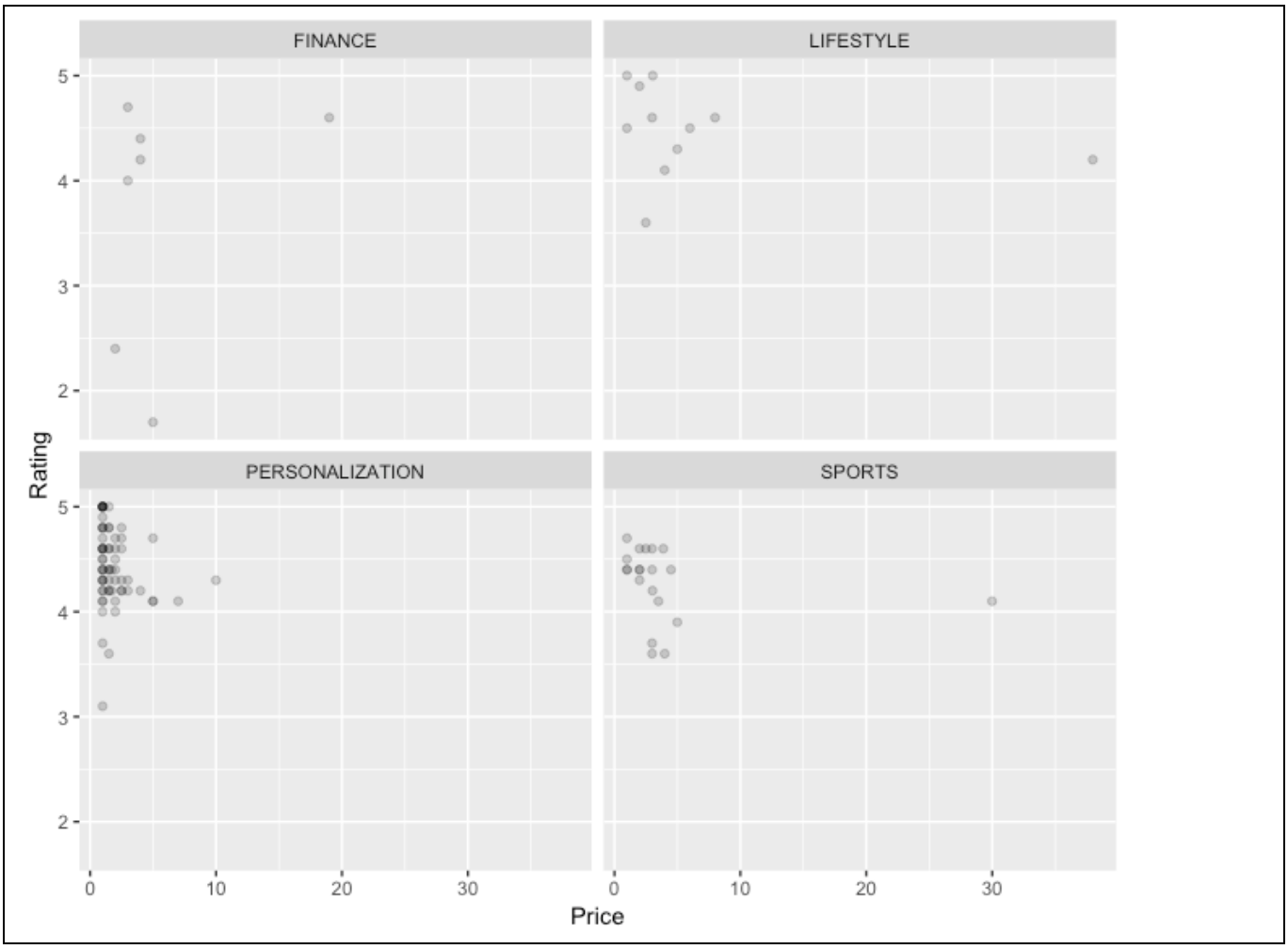
1M byte 이하의 앱은 가격이 대부분 10$ 미만이지만, 많은 앱이 분포하고 있는 1M byte ~ 100 Mbyte 구간에서는 가격도 같이 증가하고 있는 것을 볼 수 있다. 높은 가격의 앱들의 개수도 점차로 증가하였다. 단 Size 축이 로그스케일이므로, Size 의 증가에 대해 가격의 증가는 상대적으로 작게 나타난다.

(2)

그래프의 해석은 여러가지 관점에서 가능하나, 아래와 같은 해석이 무난할 것이다.
- 많은 수의 앱이 거의 $10 이하에 집중되어 있다.
- 평점 4.5 이상의 앱은 거의 $20 이하에 분포한다.
- 가격이 $20 이상인 앱은 평점이 오히려 감소한다 등.

(3)

```
>x %>% group_by(Category) %>% summarize(n=n()) %>% arrange(desc(n)) %>%
head(10)
# A tibble: 10 x 2
   Category            n
   <fct>           <int>
 1 FAMILY           1617
 2 GAME              974
 3 TOOLS             634
 4 MEDICAL           324
 5 LIFESTYLE         280
 6 PERSONALIZATION   280
 7 FINANCE           266
 8 SPORTS            247
 9 BUSINESS          246
10 PHOTOGRAPHY       236
> x %>% filter(Type=="Paid"&Price<50&Category %in% c("LIFESTYLE",
"PERSONALIZATION", "FINANCE", "SPORTS")) %>% ggplot(aes(Price,
Rating))+geom_point(alpha=0.2)+facet_wrap(~Category)
```



12 장 04 절

(1)

설명 변수 조합	모델 성능(예측값의 mse)
Category	0.2872693
Size	0.2945681

Content.Rating	0.2956962
Category + Size	0.2865305
Category + Content.Rating	0.2862756
Size + Content.Rating	0.2936191
Category + Size + Content.Rating	0.2865483

선형 회귀 모델에 의한 교차 성능보다 좋은 결과를 얻을 수 있다.

(2)

결정트리, 랜덤포리스트, SVM 을 이용한 교차검증결과에서 Category + Size 를 설명변수 조합으로 사용한 모델이 가장 성능이 좋았다. 특히 랜덤포리스트를 이용한 모델에서 가장 낮은 mse 를 보였다. 그림 12-6 의 Rating 과 Reviews 의 관계는 Reviews를 통해 (특히 높은 Reviews 의 경우에) Rating 을 일정범위에서 예측할 수 있음을 보여준다. 그러나 Reviews 는 출시 이전에 알 수 없으므로 이를 설명 변수로 사용할 수는 없음을 이미 설명한 바 있다. 그림 12-18 에 나타난 Rating 과 Size 의 관계가 그림 12-6과 유사할 뿐 아니라, 오히려 Size 에 대한 Rating의 분포가 한정되어 있어, 예측 성능은 더 좋을 수 있다. 그림 12-21 역시 Category 내에서 Rating 의 분포가 한정된 경향을 보이고 있어 예측 성능을 기대 할 수 있고, 이 두 변수의 조합된 경우 좀 더 좋은 성능이 나온 것으로 이해할 수 있다.