Homework 1

Shen Dingtao 3170104764

1. Solution:

(a) First, we load the Iowa data set into R and make it a data frame called `iowa.df` with following command.

```
iowa.df<-read.csv("data/iowa.csv",header=T,sep=";")
```

(b) Use the following command to show the dim of `iowa.df`:

```
dim(iowa.df)
```

```
## [1] 33 10
```

The result shows that `iowa.df` has 33 rows and 10 columns.

(c) Use the following command to show the names of the columns of `iowa.df`:

```
colnames(iowa.df)
```

```
##  [1] "Year"  "Rain0" "Temp1" "Rain1" "Temp2" "Rain2" "Temp3" "Rain3" "Temp4"
## [10] "Yield"
```

(d) With the following command, we can get the value of row 5, column 7 of `iowa.df` directly:

```
iowa.df[5,7]
```

```
## [1] 79.7
```

(e) Use the following command to display the second row of `iowa.df` in its entirety:

```
iowa.df[2,]
```

```
##   Year Rain0 Temp1 Rain1 Temp2 Rain2 Temp3 Rain3 Temp4 Yield
## 2 1931 14.76  57.5  3.83    75  2.72  77.2   3.3  72.6  32.9
```

2. Syntax and class-typing.

Solution:

(a) First, We try these following commands in console and get the results:

```r
vector1 <- c("5", "12", "7", "32")
max(vector1)
```

```
## [1] "7"
```

```r
sort(vector1)
```

```
## [1] "12" "32" "5"  "7"
```

```r
sum(vector1)
```

And the fourth command is error.

Explain: The first command `vector1 <- c("5", "12", "7", "32")` creates a vector of character type, not integer type. So when the parameter of `max()` and `sort()` is `vector1`, the objects to be compared are characters, that is "12" < "32" < "5" < "7", as above result shows. In this case, the function `sum()` can't be executed on character type, so the fourth command is an error.

(b)

1) For the `c()` function, The output type is determined from the highest type of the components in the hierarchy NULL < raw < logical < integer < double < complex < character < list < expression. So for the command `vector2 <-c("5",7,12)`, the type of `vector2` is character. That is

```r
(vector2 <- c("5",7,12))
```

```
## [1] "5"  "7"  "12"
```

Thus, `vector2[2]="7"`, `vector2[3]="12"`, they are not numeric, which means `vector2[2]+vector2[3]` is an error. 2) The function `data.frame` create data frames:

```r
dataframe3 <- data.frame(z1="5",z2=7,z3=12)
dataframe3
```

```
##   z1 z2 z3
## 1  5  7 12
```

So the first row gives the value of variables `z1,z2,z3`, and `dataframe3[1,2]=7,dataframe3[1,3]=12`, thus

```r
dataframe3[1,2]+dataframe3[1,3]
```

```
## [1] 19
```

3) The function `list()` returns a list or dotted pair list composed of its arguments with each value either tagged or untagged. Thus,

```r
list4 <- list(z1="6", z2=42, z3="49", z4=126)
list4
```

```
## $z1
## [1] "6"
##
## $z2
## [1] 42
##
## $z3
## [1] "49"
##
## $z4
## [1] 126
```

And to access elements of the list, `[[ ]]` drops the names and structures, `[ ]` doesn't. That is

```
list4[[2]]
```

```
## [1] 42
```

```
list4[[4]]
```

```
## [1] 126
```

They are numeric.

```
list4[2]
```

```
## $z2
## [1] 42
```

```
list4[4]
```

```
## $z4
## [1] 126
```

They are not numberic. Then `list4[[2]]+list4[[4]]` is legal and `list4[2]+list4[4]` is an error.

```
list4[[2]]+list4[[4]]
```

```
## [1] 168
```

3. Working with functions and operators.

Solution:

(a) Use following command to create the sequence of numbers from 1 to 10000 in increments of 372.

```
seq1=seq(1,10000,by = 372)
seq1
```

```
##  [1]    1  373  745 1117 1489 1861 2233 2605 2977 3349 3721 4093 4465 4837 5209
## [16] 5581 5953 6325 6697 7069 7441 7813 8185 8557 8929 9301 9673
```

Use the following command to create a sequence between 1 and 10000 that is exactly 50 numbers in length.

```r
seq2=seq(1,10000,length.out=50)
seq2
```

```
##  [1]     1.0000   205.0612   409.1224   613.1837   817.2449  1021.3061
##  [7]  1225.3673  1429.4286  1633.4898  1837.5510  2041.6122  2245.6735
## [13]  2449.7347  2653.7959  2857.8571  3061.9184  3265.9796  3470.0408
## [19]  3674.1020  3878.1633  4082.2245  4286.2857  4490.3469  4694.4082
## [25]  4898.4694  5102.5306  5306.5918  5510.6531  5714.7143  5918.7755
## [31]  6122.8367  6326.8980  6530.9592  6735.0204  6939.0816  7143.1429
## [37]  7347.2041  7551.2653  7755.3265  7959.3878  8163.4490  8367.5102
## [43]  8571.5714  8775.6327  8979.6939  9183.7551  9387.8163  9591.8776
## [49]  9795.9388 10000.0000
```

(b) `rep(1:3, times=3)` repeats the whole vector `1:3` for three times, that is

```r
rep(1:3, times=3)
```

```
## [1] 1 2 3 1 2 3 1 2 3
```

While `rep(1:3, each=3)` repeats each element of the vector `1:3` for 3 times. That is

```r
rep(1:3, each=3)
```

```
## [1] 1 1 1 2 2 2 3 3 3
```

MB.Ch1.2. Create a new data frame `part_orings` by extracting these rows from orings

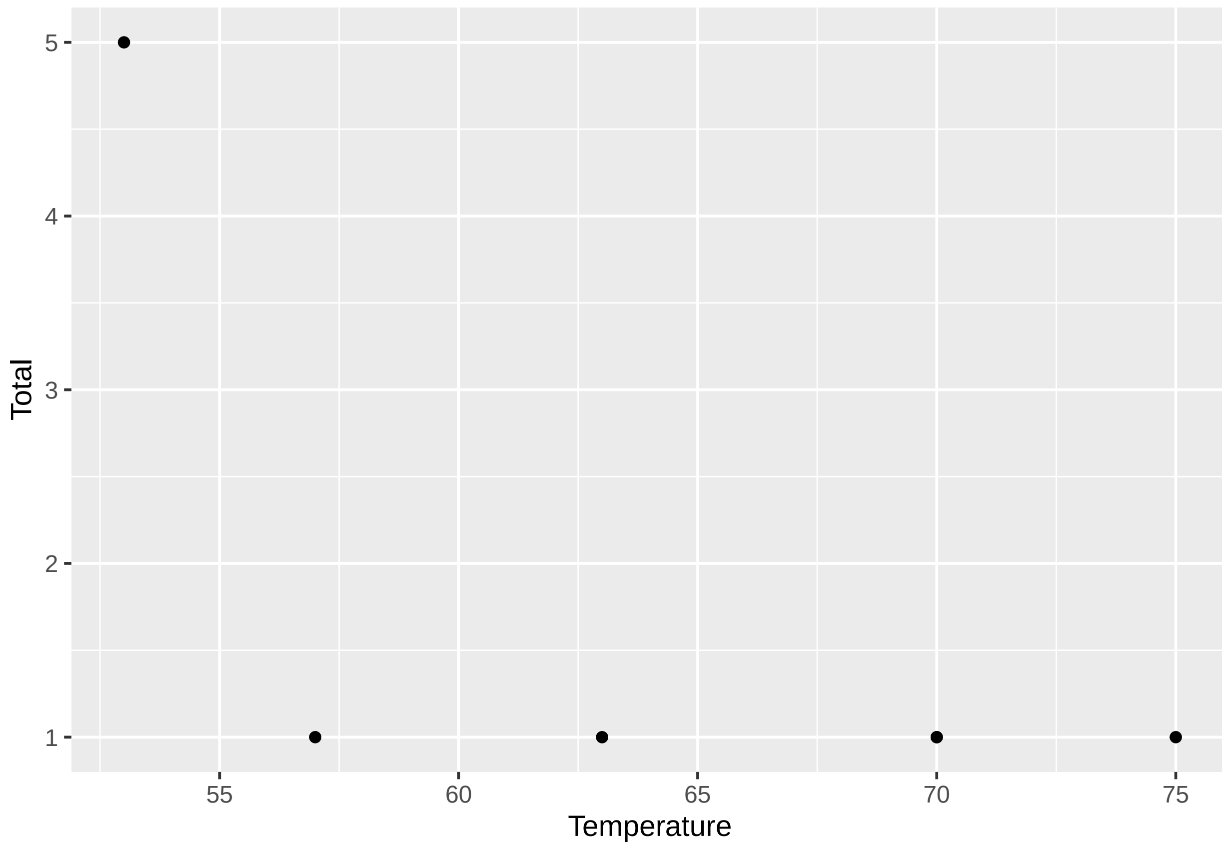
```r
part_orings<-orings[c(1,2,4,11,13,18),]
```

```r
part_orings
```

```
##    Temperature Erosion Blowby Total
## 1           53       3      2     5
## 2           57       1      0     1
## 4           63       1      0     1
## 11          70       1      0     1
## 13          70       1      0     1
## 18          75       0      2     1
```

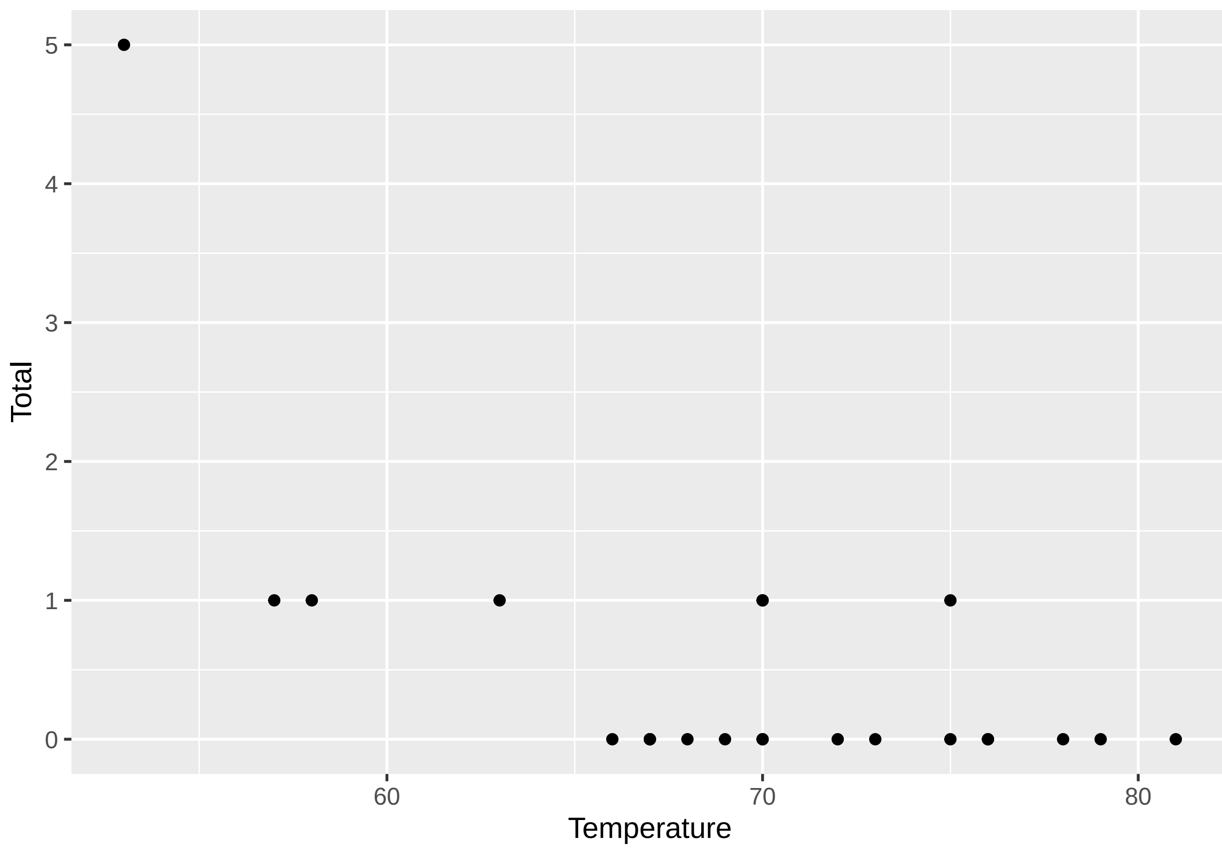
Plot total incidents against temperature for this new data frame:

```r
ggplot(data = part_orings) +
     geom_point(aes(x = Temperature, y = Total))
```



Plot total incidents against temperature for the full data set:

```
ggplot(data = orings) +
    geom_point(aes(x = Temperature, y = Total))
```



MB.Ch1.4. For the data frame ais (DAAG package)

(a) Use the function str() to get information on each of the columns:

```
str(ais)
```

```
## 'data.frame':    202 obs. of  13 variables:
##  $ rcc    : num  3.96 4.41 4.14 4.11 4.45 4.1 4.31 4.42 4.3 4.51 ...
##  $ wcc    : num  7.5 8.3 5 5.3 6.8 4.4 5.3 5.7 8.9 4.4 ...
##  $ hc     : num  37.5 38.2 36.4 37.3 41.5 37.4 39.6 39.9 41.1 41.6 ...
##  $ hg     : num  12.3 12.7 11.6 12.6 14 12.5 12.8 13.2 13.5 12.7 ...
##  $ ferr   : num  60 68 21 69 29 42 73 44 41 44 ...
##  $ bmi    : num  20.6 20.7 21.9 21.9 19 ...
##  $ ssf    : num  109.1 102.8 104.6 126.4 80.3 ...
##  $ pcBfat: num  19.8 21.3 19.9 23.7 17.6 ...
##  $ lbm    : num  63.3 58.5 55.4 57.2 53.2 ...
##  $ ht     : num  196 190 178 185 185 ...
##  $ wt     : num  78.9 74.4 69.1 74.9 64.6 63.7 75.2 62.3 66.5 62.9 ...
##  $ sex    : Factor w/ 2 levels "f","m": 1 1 1 1 1 1 1 1 1 1 ...
##  $ sport : Factor w/ 10 levels "B_Ball","Field",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Use `complete.case()` to determine the rows in which one or more values is missing.

```
complete.cases(ais)
```

```
##   [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
##  [16] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [31] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [46] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [61] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [76] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
##  [91] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [106] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [121] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [136] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [151] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [166] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [181] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
## [196] TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

The result shows that there is no missing value.

(b) There are two methods to make a table that shows the numbers of males and females for each different sport. Solution 1:

```r
new_ais1<-table(ais[,13],ais[,12])
```

```r
new_ais1
```

```
## 
##            f  m
##   B_Ball  13 12
##   Field    7 12
##   Gym      4  0
##   Netball 23  0
##   Row     22 15
##   Swim     9 13
##   T_400m  11 18
##   T_Sprnt  4 11
##   Tennis   7  4
##   W_Polo   0 17
```

Solution 2:

```r
new_ais2 <- ais %>%
  group_by(sport) %>%
  count(sex)
(new_ais<-spread(new_ais2,sex,n))
```

```
## # A tibble: 10 x 3
## # Groups:   sport [10]
##    sport       f     m
##    <fct>   <int> <int>
##  1 B_Ball     13    12
##  2 Field       7    12
##  3 Gym         4    NA
##  4 Netball    23    NA
##  5 Row        22    15
##  6 Swim        9    13
```

```
##   7 T_400m      11    18
##   8 T_Sprnt      4    11
##   9 Tennis       7     4
##  10 W_Polo      NA    17
```

To determine if there is a large imbalance (e.g., by a factor of more than 2:1 or less than 1:2) in the numbers of the two sexes, we add a column `fac` to show the factor `f/m`:

```
new_ais %>% mutate(fac=f/m)
```

```
## # A tibble: 10 x 4
## # Groups:   sport [10]
##     sport       f     m    fac
##     <fct>   <int> <int>  <dbl>
##   1 B_Ball     13    12  1.08
##   2 Field       7    12  0.583
##   3 Gym         4    NA NA
##   4 Netball    23    NA NA
##   5 Row        22    15  1.47
##   6 Swim        9    13  0.692
##   7 T_400m     11    18  0.611
##   8 T_Sprnt     4    11  0.364
##   9 Tennis      7     4  1.75
##  10 W_Polo     NA    17 NA
```

The result implies that there is a large imbalance in Gym, Netball, in which female dominates(`f/m` > 2). While in T_Sprnt and W_Polo, male dominates(`f/m` < 0.5)

MB.Ch1.6.

```
Manitoba.lakes <- data.frame(c("Winnipeg","Winnipegosis","Manitoba",
                               "SouthernIndian","Cedar","Island",
                               "Gods","Cross","Playgreen"),
                             elevation=c(217,254,248,254,253,227,178,207,217),
                             area=c(24387,5374,4624,2247,1353,1223,1151,755,657))
```

Assign the names of the lakes using the row.names() function:

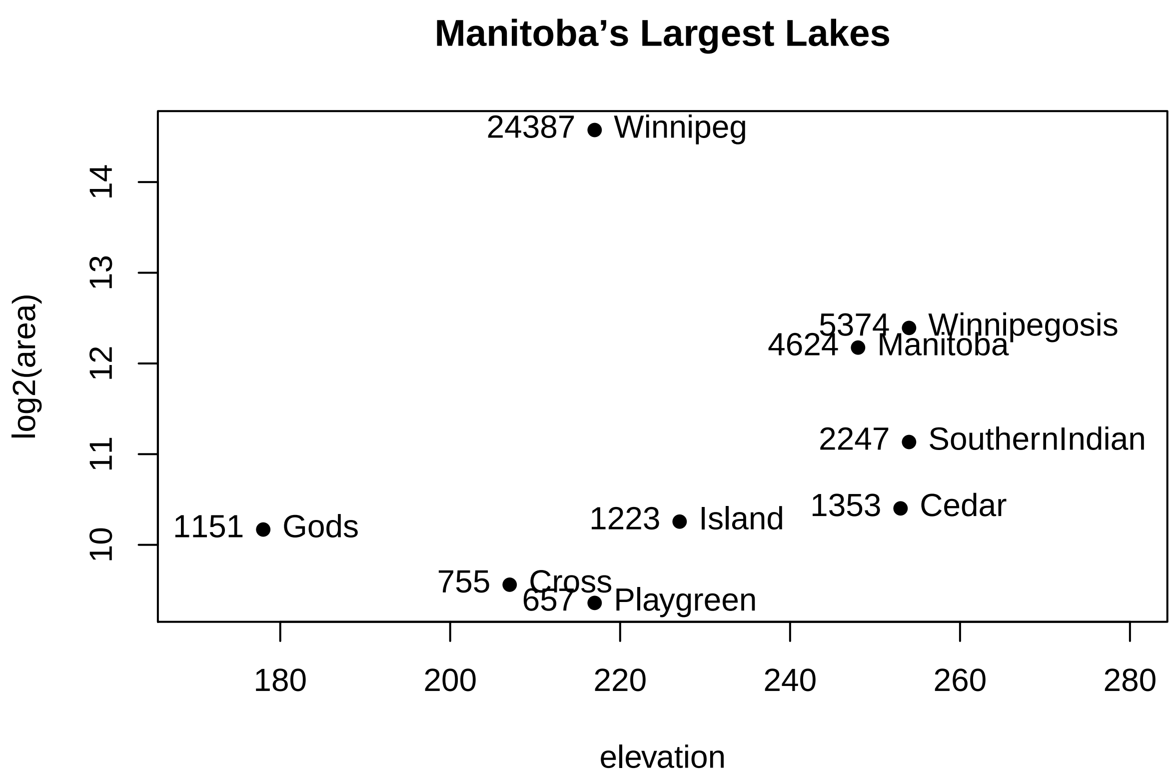
```
row.names(Manitoba.lakes)<-Manitoba.lakes[,1]
```

```
rownames(Manitoba.lakes)
```

```
## [1] "Winnipeg"       "Winnipegosis"   "Manitoba"       "SouthernIndian"
## [5] "Cedar"          "Island"         "Gods"           "Cross"
## [9] "Playgreen"
```

(a) Use the following code to plot log2(area) versus elevation, adding labeling information (there is an extreme value of area that makes a logarithmic scale pretty much essential):

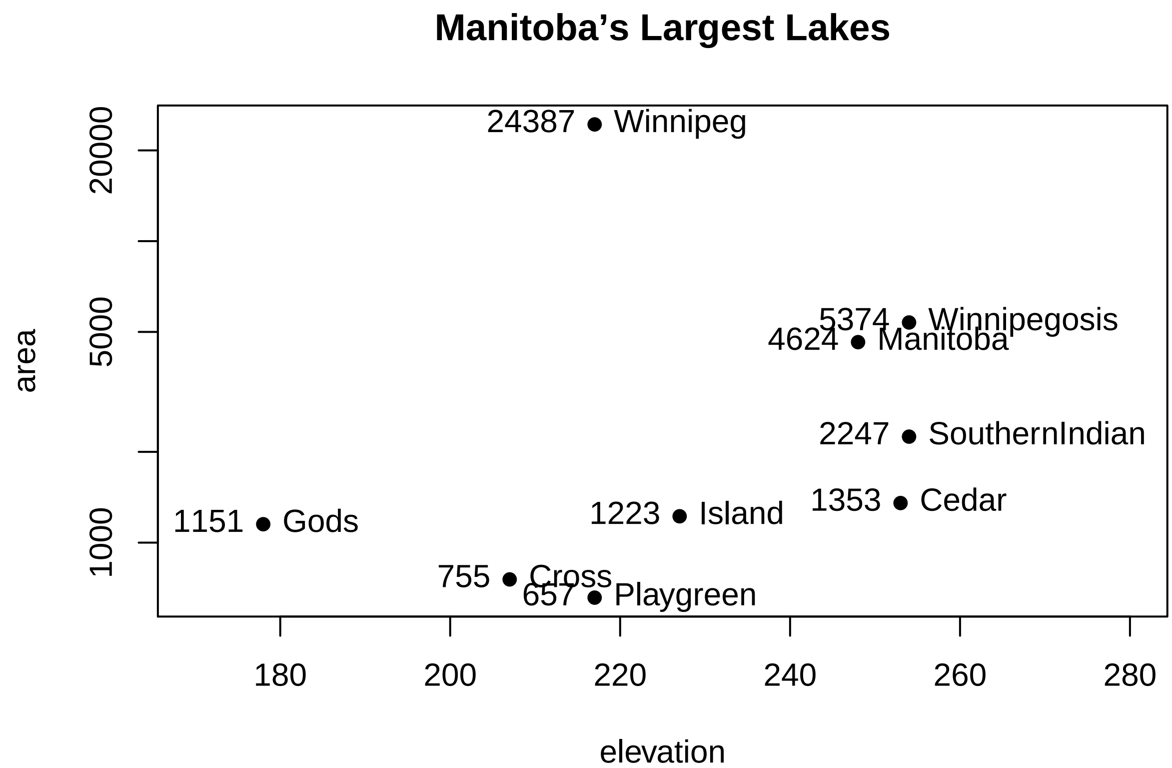
```r
attach(Manitoba.lakes)
plot(log2(area) ~ elevation, pch=16, xlim=c(170,280))
# NB: Doubling the area increases log2(area) by 1.0
text(log2(area) ~ elevation, labels=row.names(Manitoba.lakes), pos=4)
text(log2(area) ~ elevation, labels=area, pos=2)
title("Manitoba's Largest Lakes")
```



Captions: the labeling on y-axis gives a logarithmic scale that shows the logs base 2 of area. And the labeling on the points gives the corresponding name of the lake and the actual area of the lake. In this way, the scale on the y-axis will increase by one unit, that is 1, while the area is doubled.

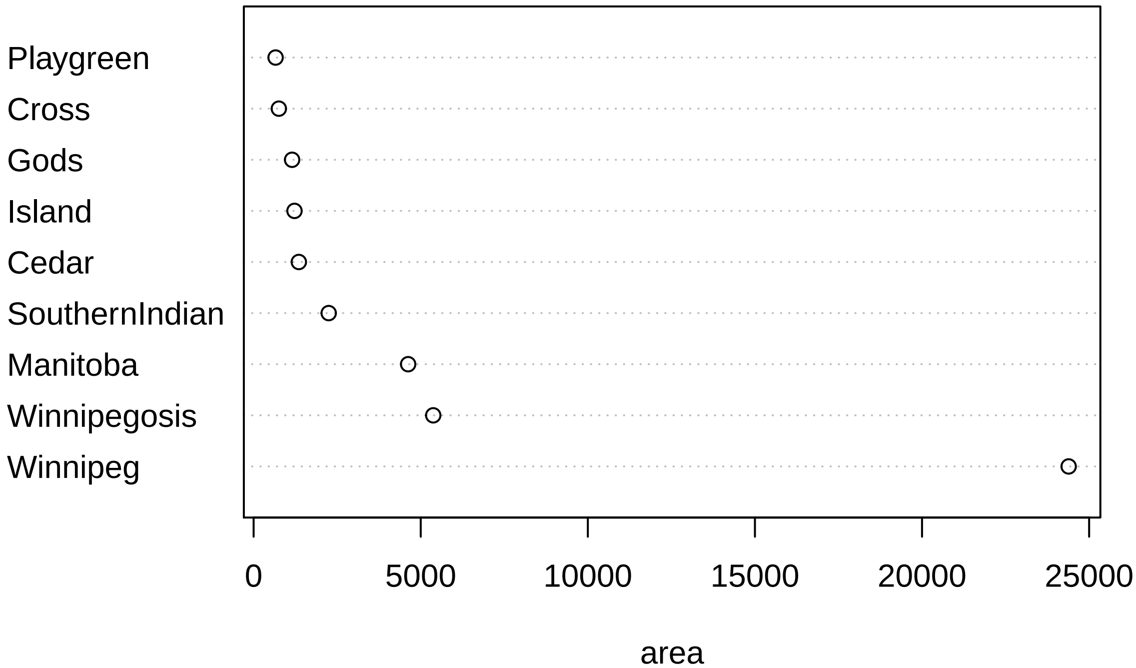
(b) Repeat the plot and associated labeling, now plotting area versus elevation, but specifying log="y" in order to obtain a logarithmic y-scale.

```r
plot(area ~ elevation, pch=16, xlim=c(170,280), log="y")
text(area ~ elevation, labels=row.names(Manitoba.lakes), pos=4, ylog=T)
text(area ~ elevation, labels=area, pos=2, ylog=T)
title("Manitoba's Largest Lakes")
```



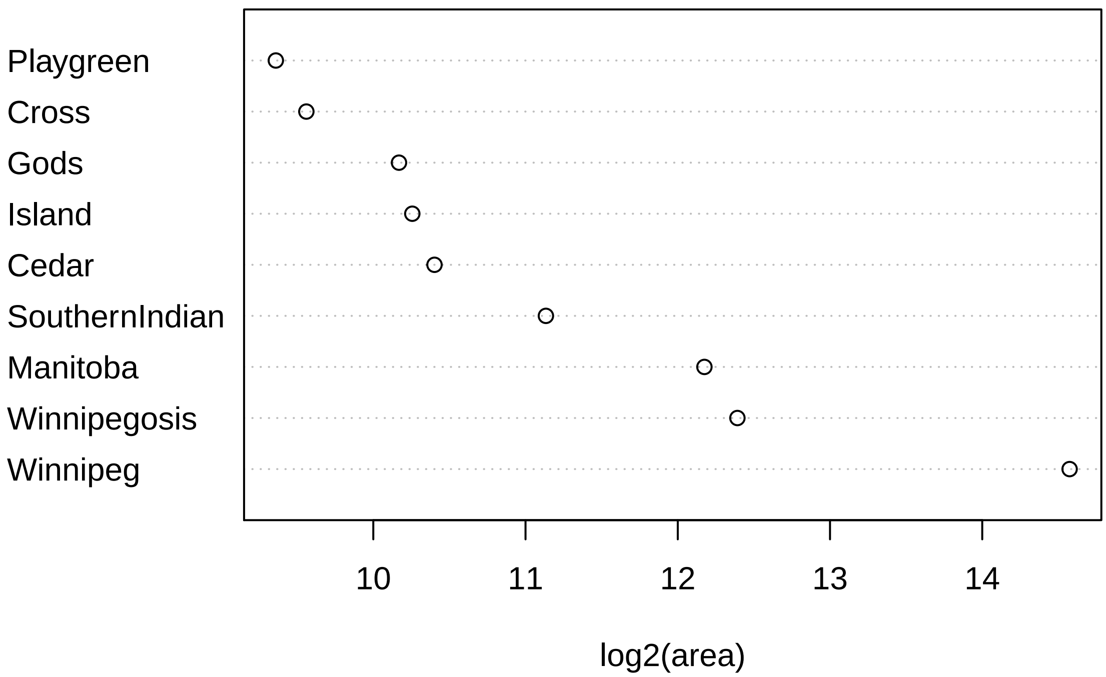
MB.Ch1.7. (a)

```r
dotchart(area,labels=Manitoba.lakes[,1],xlab="area")
```



(b)

```
dotchart(log2(area),labels=Manitoba.lakes[,1],xlab="log2(area)")
```



MB.Ch1.8. The lower bound for the area of Manitoba covered by water is just the whole area of all lakes in Manitoba. That is

```
sum(Manitoba.lakes$area)
```

```
## [1] 41771
```