Homework 1 - Report

3170105743 李政达

2020/7/7

Question 1

The Iowa data set iowa.csv is a toy example that summarises the yield of wheat (bushels per acre) for the state of Iowa between 1930-1962. In addition to yield, year, rainfall and temperature were recorded as the main predictors of yield.

a. First, we need to load the data set into R using the command read.csv() . Use the help function to learn what arguments this function takes. Once you have the necessary input, load the data set into R and make it a data frame called iowa.df.

Ans: We can use help("read.csv") to learn the parameters of the function. The first parameter is the file name, and it may change in different case. I save the file in the data folder, which is in the parent directory of this .rmd file, so I set the first parameter to `"../data/iowa.csv"`. `header = TRUE` means that the first row contains the names of the variables. `sep = ";"` means that the field separator is semicolon.

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

```
##    Year Rain0 Temp1 Rain1 Temp2 Rain2 Temp3 Rain3 Temp4 Yield
## 1  1930 17.75  60.2  5.83  69.0  1.49  77.9  2.42  74.4  34.0
## 2  1931 14.76  57.5  3.83  75.0  2.72  77.2  3.30  72.6  32.9
## 3  1932 27.99  62.3  5.17  72.0  3.12  75.8  7.10  72.2  43.0
## 4  1933 16.76  60.5  1.64  77.8  3.45  76.4  3.01  70.5  40.0
## 5  1934 11.36  69.5  3.49  77.2  3.85  79.7  2.84  73.4  23.0
## 6  1935 22.71  55.0  7.00  65.9  3.35  79.4  2.42  73.6  38.4
## 7  1936 17.91  66.2  2.85  70.1  0.51  83.4  3.48  79.2  20.0
## 8  1937 23.31  61.8  3.80  69.0  2.63  75.9  3.99  77.8  44.6
## 9  1938 18.53  59.5  4.67  69.2  4.24  76.5  3.82  75.7  46.3
## 10 1939 18.56  66.4  5.32  71.4  3.15  76.2  4.72  70.7  52.2
## 11 1940 12.45  58.4  3.56  71.3  4.57  76.7  6.44  70.7  52.3
## 12 1941 16.05  66.0  6.20  70.0  2.24  75.1  1.94  75.1  51.0
## 13 1942 27.10  59.3  5.93  69.7  4.89  74.3  3.17  72.2  59.9
## 14 1943 19.05  57.5  6.16  71.6  4.56  75.4  5.07  74.0  54.7
## 15 1944 20.79  64.6  5.88  71.7  3.73  72.6  5.88  71.8  52.0
## 16 1945 21.88  55.1  4.70  64.1  2.96  72.1  3.43  72.5  43.5
## 17 1946 20.02  56.5  6.41  69.8  2.45  73.8  3.56  68.9  56.7
## 18 1947 23.17  55.6 10.39  66.3  1.72  72.8  1.49  80.6  30.5
## 19 1948 19.15  59.2  3.42  68.6  4.14  75.0  2.54  73.9  60.5
## 20 1949 18.28  63.5  5.51  72.4  3.47  76.2  2.34  73.0  46.1
## 21 1950 18.45  59.8  5.70  68.4  4.65  69.7  2.39  67.7  48.2
## 22 1951 22.00  62.2  6.11  65.2  4.45  72.1  6.21  70.5  43.1
## 23 1952 19.05  59.6  5.40  74.2  3.84  74.7  4.78  70.0  62.2
## 24 1953 15.67  60.0  5.31  73.2  3.28  74.6  2.33  73.2  52.9
## 25 1954 15.92  55.6  6.36  72.9  1.79  77.4  7.10  72.1  53.9
## 26 1955 16.75  63.6  3.07  67.2  3.29  79.8  1.79  77.2  48.4
## 27 1956 12.34  62.4  2.56  74.7  4.51  72.7  4.42  73.0  52.8
## 28 1957 15.82  59.0  4.84  68.9  3.54  77.9  3.76  72.9  62.1
## 29 1958 15.24  62.5  3.80  66.4  7.55  70.5  2.55  73.0  66.0
## 30 1959 21.72  62.8  4.11  71.5  2.29  72.3  4.92  76.3  64.2
## 31 1960 25.08  59.7  4.43  67.4  2.76  72.6  5.36  73.2  63.2
## 32 1961 17.79  57.4  3.36  69.4  5.51  72.6  3.04  72.4  75.4
## 33 1962 26.61  66.6  3.12  69.1  6.27  71.6  4.31  72.5  76.0
```

b. How many rows and columns does iowa.df have?

```
length(row.names(iowa.df))
```

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

```
length(names(iowa.df))
```

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

c. What are the names of the columns of iowa.df?

```
names(iowa.df)
```

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

d. What is the value of row 5, column 7 of iowa.df?

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

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

e. 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
```

Question 2

Syntax and class-typing.

a. For each of the following commands, either explain why they should be errors, or explain the non-erroneous result.

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

Ans:

- `c()` can combine values into a vector or list, so `vector1` will be a vector containing `"5"  "12" "7" "32"`.
- `max()` return the maximum of the input values. The type of input is character, which can be compare. So the command runs correctly and returns the maximum, which is `"7"`.
- `sort()` can sort a vector into ascending order. The type of input is `character`, which can be compare. So the command runs correctly and sorts the vector, and outputs `"12" "32" "5"  "7"`.

- `sum()` can compute the sums of all the input values, but the character type can not be added, so the command will cause an error.

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

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

```
sort(vector1)
```

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

```
#sum(vector1) #error
```

b. For the next series of commands, either explain their results, or why they should produce errors.

```
vector2 <- c("5",7,12)
vector2[2] + vector2[3]

dataframe3 <- data.frame(z1="5",z2=7,z3=12)
dataframe3[1,2] + dataframe3[1,3]

list4 <- list(z1="6", z2=42, z3="49", z4=126)
list4[[2]]+list4[[4]]
list4[2]+list4[4]
```

Ans:

- `c()` will combine all kinds of values, but the type of the values may change in order to compatibility. For example, Boolean variable will change its type when combined with integer variable, and integer variable will change its type when combined with double variable. In this command, the type of the elements in `vector2` will be character. Because the values of `vector2` is character, they can not be added. So the command will cause an error.
- `data.frame()` creates a data frame, and we can access the element at row `x` and column `y` by `dataframe3[x, y]`. Because `dataframe3[1, 2] = 7` and `dataframe3[1, 3] = 12`, the command will output 19.
- We can use `[]` and `[[]]` to access the elements in a list. `[[]]` drops names and structures, but `[]` does not. So `list4[[2]]+list4[[4]] = 42+126 = 168`, but `list4[2]+list4[4]` causes an error.

```
vector2 <- c("5",7,12)
#vector2[2] + vector2[3] #error

dataframe3 <- data.frame(z1="5",z2=7,z3=12)
dataframe3[1,2] + dataframe3[1,3]
```

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

```
list4 <- list(z1="6", z2=42, z3="49", z4=126)
list4[[2]]+list4[[4]]
```

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

```
#list4[2]+list4[4] #error
```

Question 3

Working with functions and operators.

a. The colon operator will create a sequence of integers in order. It is a special case of the function `seq()` which you saw earlier in this assignment. Using the help command `?seq` to learn about the function, design an expression that will give you the sequence of numbers from 1 to 10000 in increments of 372. Design another that will give you a sequence between 1 and 10000 that is exactly 50 numbers in length.

Ans: From the help documentation, we can learn that `seq()` function accept many parameters. For example, the parameter `by` is the increment of the sequence, and the parameter `length.out` is the desired length of the sequence.

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

```
##  [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
```

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

```
##  [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. The function `rep()` repeats a vector some number of times. Explain the difference between `rep(1:3, times=3)` and `rep(1:3, each=3)`.

Ans: The parameter `times` is the number of the times that the whole vector is repeated, and the parameter `each` is the number of the times that each element of the input vector.

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

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

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

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

MB.Ch1.2

The orings data frame gives data on the damage that had occurred in US space shuttle launches prior to the disastrous Challenger launch of 28 January 1986. The observations in rows 1, 2, 4, 11, 13, and 18 were included in the pre-launch charts used in deciding whether to proceed with the launch, while remaining rows were omitted.

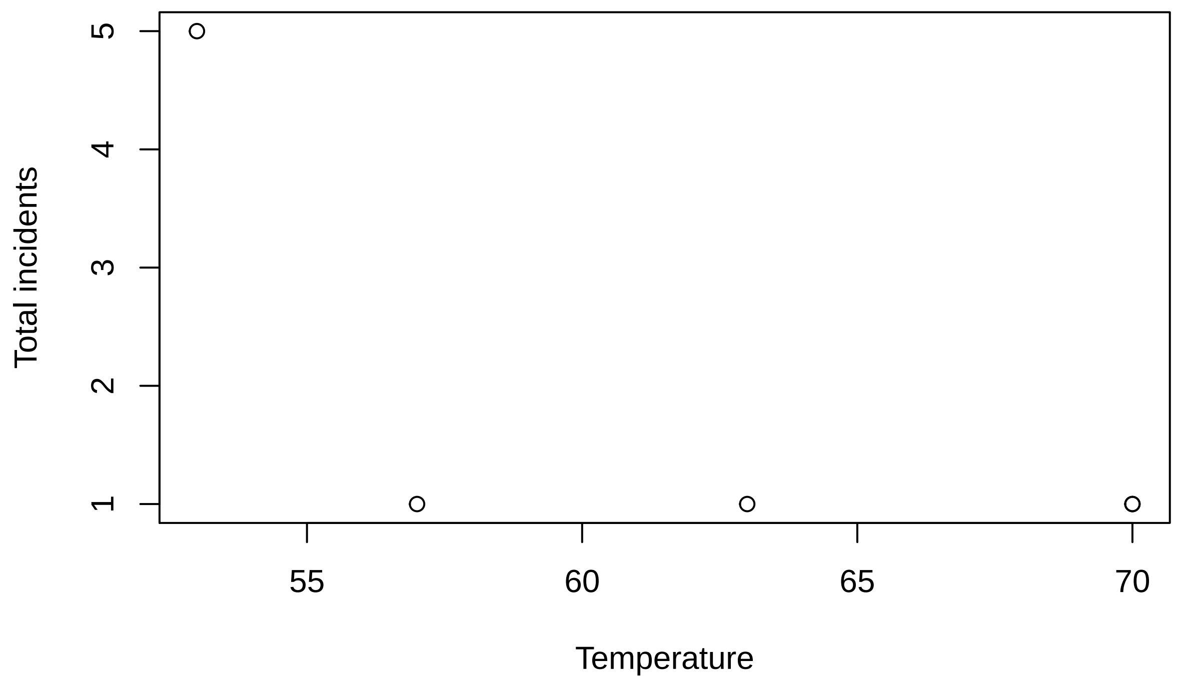
Create a new data frame by extracting these rows from orings, and plot total incidents against temperature for this new data frame. Obtain a similar plot for the full data set.

Ans: We can load the `DAAG` package and get the orings data frame. Before we load `DAAG` package, we should load `lattice` package in advance, but RStudio can do it automatically. We can use `plot()` function to plot the figures asked.

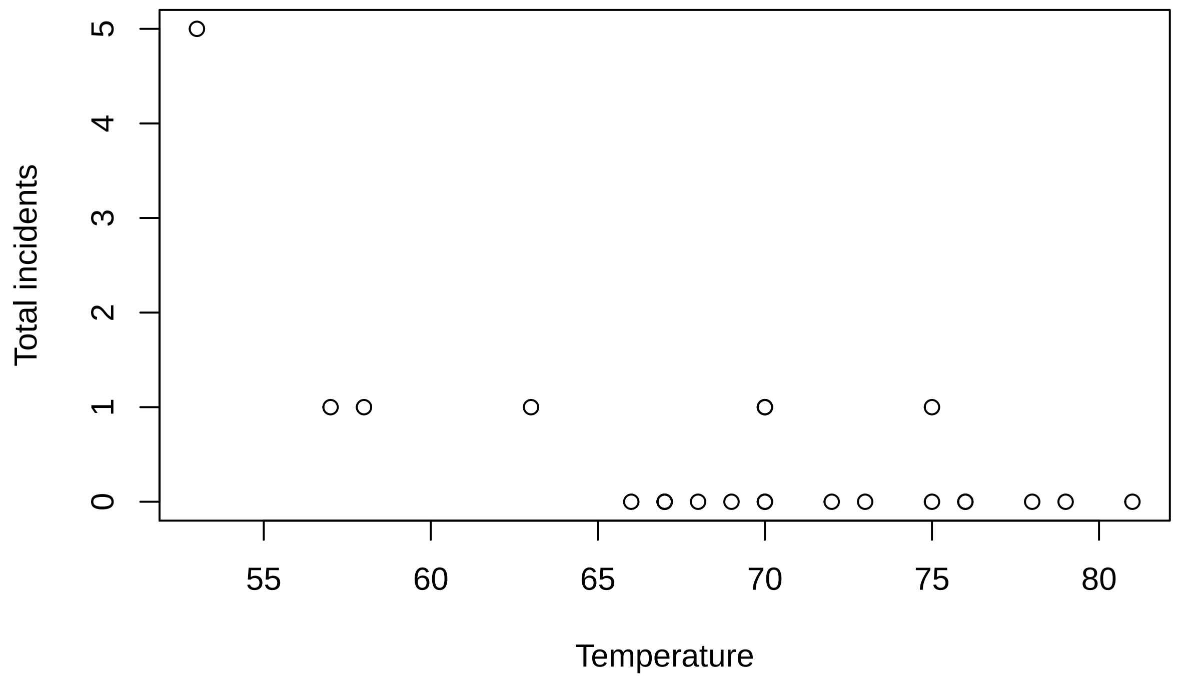
```r
#library(lattice) #RStudio can also do this step
library(DAAG)
```

```
## Loading required package: lattice
```

```
orings <- DAAG::orings
dataframe1 <- orings[c(1, 2, 4, 11, 13), ]
# for this new data frame
plot(dataframe1[, 1], dataframe1[, 4],
     xlab = "Temperature", ylab = "Total incidents")
```



```
# for the full data set
plot(orings[, 1], orings[, 4],
     xlab = "Temperature", ylab = "Total incidents")
```



MB.Ch1.4

For the data frame ais (DAAG package)

(a) Use the function str() to get information on each of the columns. Determine whether any of the columns hold missing values.

Ans: The function `is.na()` can tell that if a variable is NA. And `is.na(ais)` can tell that if an element is missing. Then we can use `sum(is.na(ais))` to count the missing values in the `ais`. In this problem, `sum(is.na(ais)) = 0`, which means that there is no missing value in any columns. If we want to check if a column holds missing values, we can use `sum(is.na(ais$column))`.

```
ais <- DAAG::ais
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 ...
```

```r
# determine whether any of the columns hold missing values
sum(is.na(ais))
```

```
## [1] 0
```

(b) Make a table that shows the numbers of males and females for each different sport. In which sports is there a large imbalance (e.g., by a factor of more than 2:1) in the numbers of the two sexes?

```r
# make a table
table(ais$sex, ais$sport)
```

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

MB.Ch1.6

Create a data frame called Manitoba.lakes that contains the lake's elevation (in meters above sea level) and area (in square kilometers) as listed below. Assign the names of the lakes using the row.names() function. elevation area Winnipeg 217 24387 Winnipegosis 254 5374 Manitoba 248 4624 SouthernIndian 254 2247 Cedar 253 1353 Island 227 1223 Gods 178 1151 Cross 207 755 Playgreen 217 657

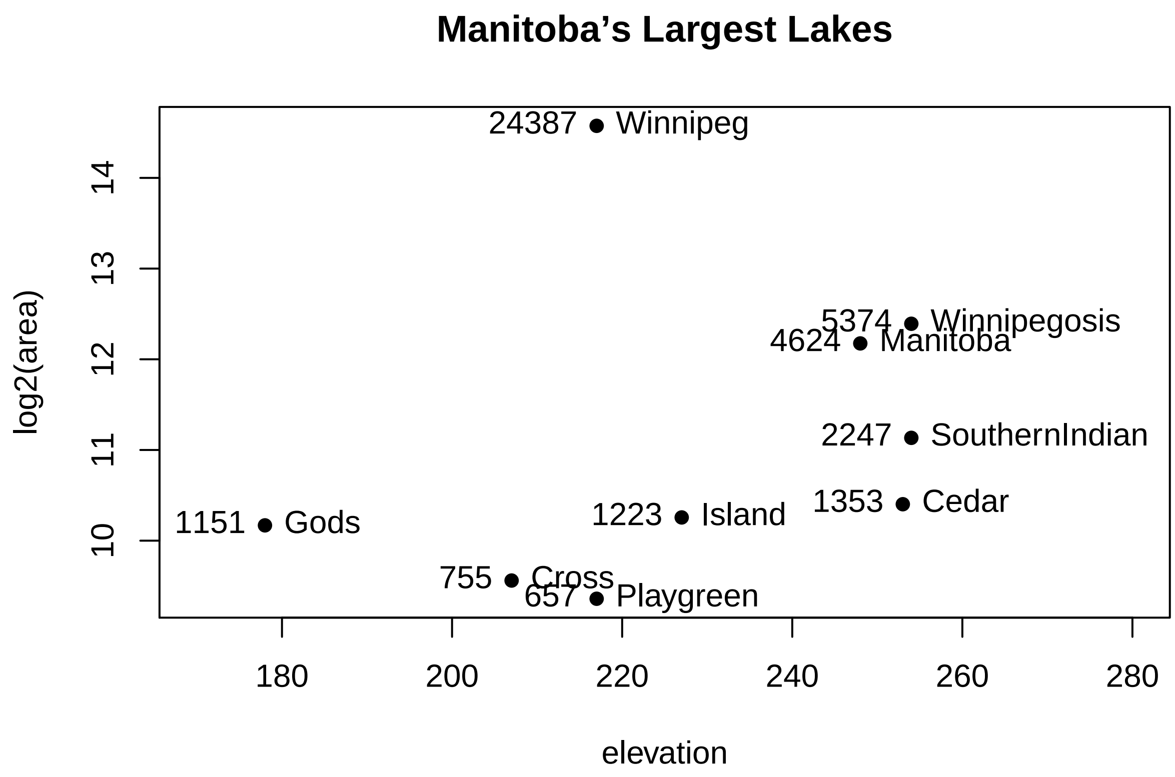
```r
elevation <- c(217, 254, 248, 254, 253, 227, 178, 207, 217)
area <- c(24387, 5374, 4624, 2247, 1353, 1223, 1151, 755, 657)
names <- c("Winnipeg", "Winnipegosis", "Manitoba", "SouthernIndian",
           "Cedar", "Island", "Gods", "Cross", "Playgreen")
Manitoba.lakes <- data.frame("elevation" = elevation,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

(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)
```

```
## The following objects are masked _by_ .GlobalEnv:
##
##     area, elevation
```

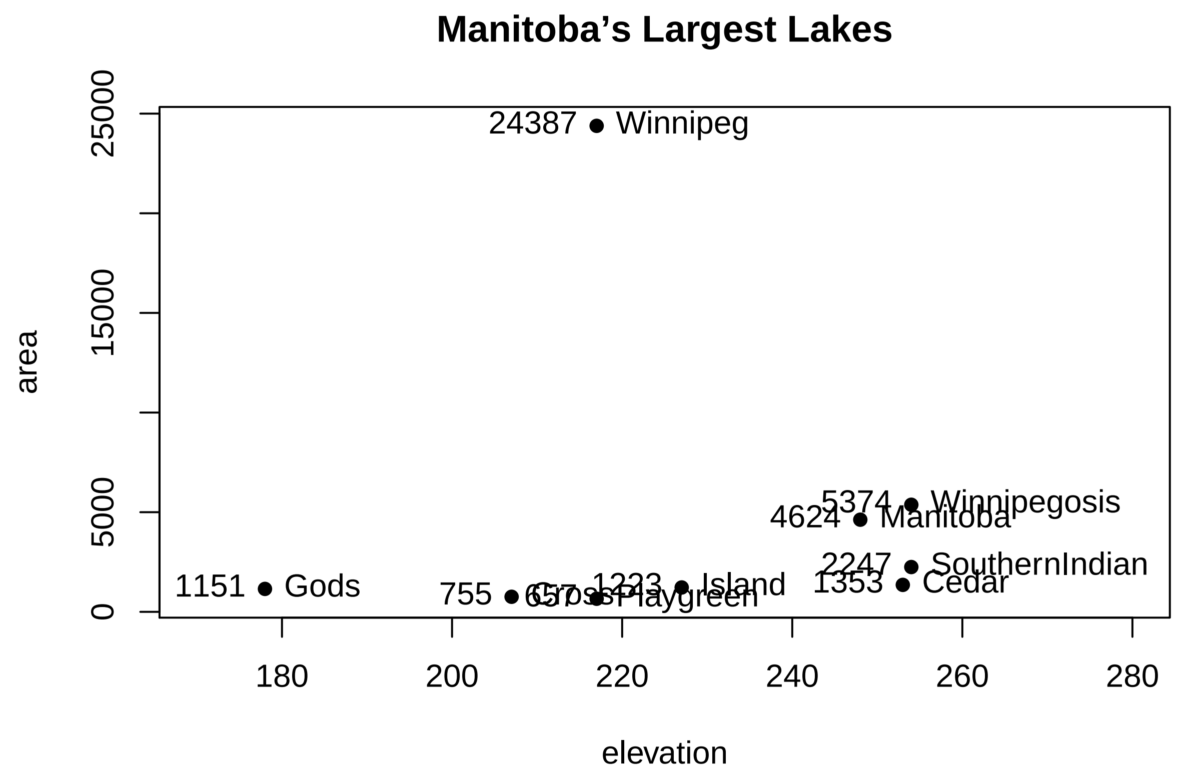
```r
plot(log2(area) ~ elevation, pch=16, xlim=c(170,280))
# NB: Doubling the area increases log2(area) by 1.0
text(log2(area) ~ elevation, labels=row.names(Manitoba.lakes), pos=4)
text(log2(area) ~ elevation, labels=area, pos=2)
title("Manitoba’ s Largest Lakes")
```



Devise captions that explain the labeling on the points and on the y-axis. It will be necessary to explain how distances on the scale relate to changes in area.

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

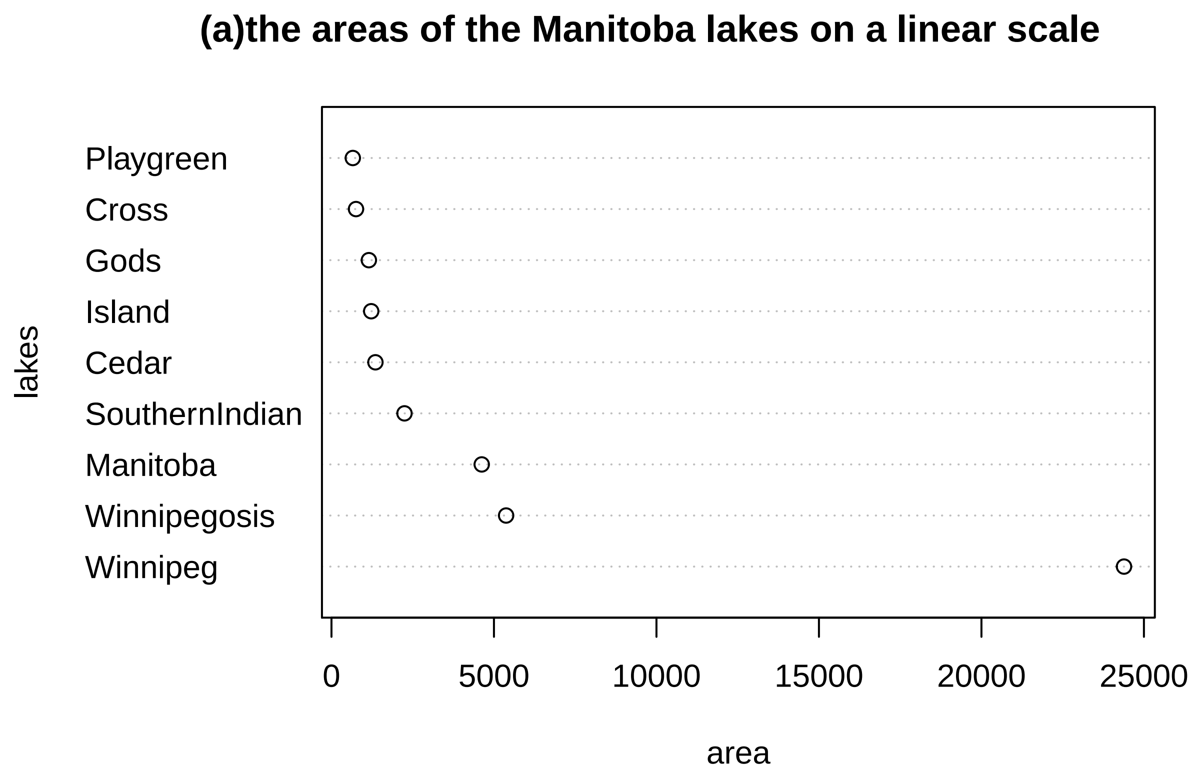
```
plot(area ~ elevation, pch=16, xlim=c(170,280), ylog=T)
text(area ~ elevation, labels=row.names(Manitoba.lakes), pos=4, ylog=T)
text(area ~ elevation, labels=area, pos=2, ylog=T)
title("Manitoba’ s Largest Lakes")
```



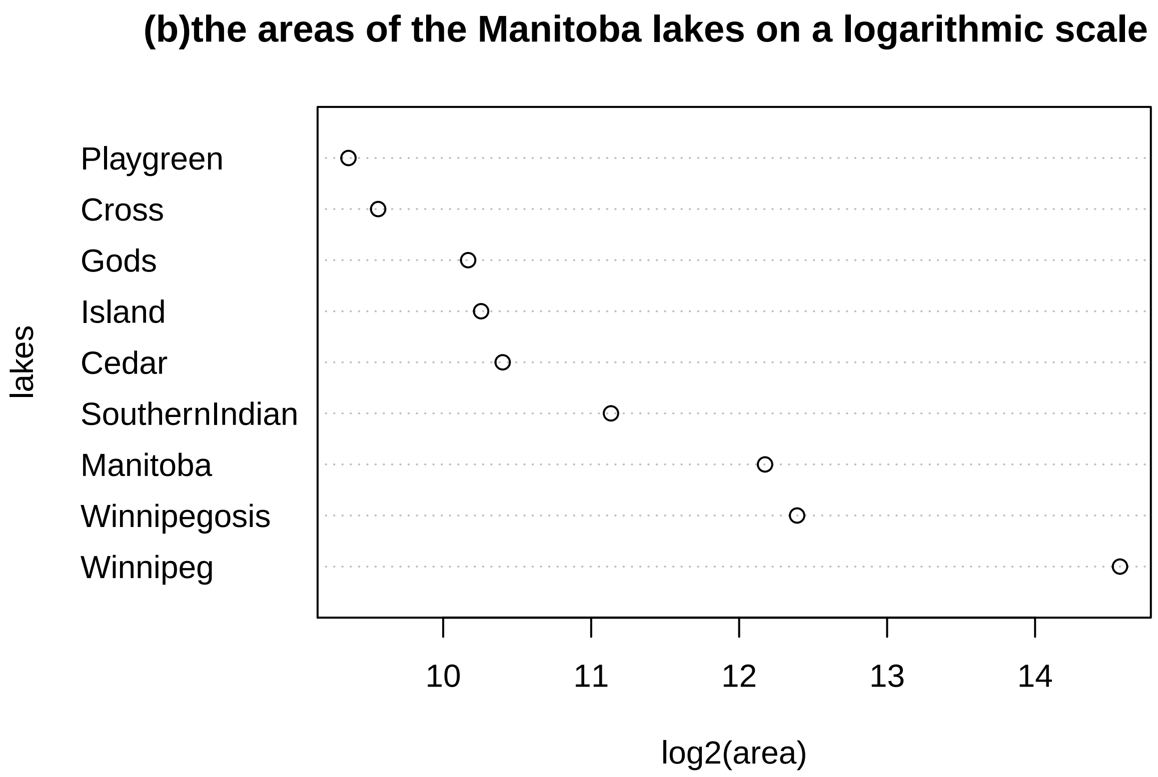
MB.Ch1.7

Look up the help page for the R function `dotchart()`. Use this function to display the areas of the Manitoba lakes (a) on a linear scale, and (b) on a logarithmic scale. Add, in each case, suitable labeling information.

```
# (a)
dotchart(area, xlab = "area", ylab = "lakes", labels = names)
title("(a)the areas of the Manitoba lakes on a linear scale")
```



```r
# (b)
dotchart(log2(area), xlab = "log2(area)", ylab = "lakes", labels = names)
title("(b)the areas of the Manitoba lakes on a logarithmic scale")
```



MB.Ch1.8

Using the sum() function, obtain a lower bound for the area of Manitoba covered by water.

```
sum(area)
```

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