

# Homework 1

Xinran Ren, 3220103492

2025/06/26

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`.

```
?read.csv
iowa.df<-read.csv("data/iowa.csv", sep = ';', header=T)
```

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

```
dim(iowa.df)
## [1] 33 10

# or
nrow(iowa.df)
## [1] 33

ncol(iowa.df)
## [1] 10
```

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

```
colnames(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
```

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)
```

- `max(vector1)`:
  - Result: "7"
  - Explanation: While the elements appear numeric, they're stored as character strings. R performs

lexicographical comparison: Compares first characters: '7' > '5' > '3' > '1'. Thus "7" is considered the "maximum" string

- `sort(vector1)`:
  - Result: "12" "32" "5" "7"
  - Explanation: Character strings are sorted alphabetically by Unicode values: '1' (ASCII 49) < '3' (51) < '5' (53) < '7' (55) Note this differs from numerical sorting (which would be 5,7,12,32)
- `sum(vector1)`:
  - Error in `sum(vector1)` : invalid 'type' (character) of argument
  - Explanation: The `sum()` function requires numeric input. No implicit type conversion occurs, unlike some other languages.

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]
```

- `vector2[2] + vector2[3]`:
  - Result: Error in `vector2[2] + vector2[3]` : non-numeric argument to binary operator
  - Explanation: We created a character vector through coercion (all elements become strings) so the actual stored values: `c("5", "7", "12")`. We're trying to add two character strings "7" + "12"
- `dataframe3[1,2] + dataframe3[1,3]`:
  - Result: 19
  - Works because: Data frames maintain column types `z2` and `z3` remain numeric despite `z1` being character. We're extracting values, not columns
- `list4[[2]]+list4[[4]]`:
  - Result: 168
  - Works because: `[[ ]]` extracts actual numeric values (42 + 126). List elements maintain their original types
- `list4[2]+list4[4]`:
  - Result: Error in `list4[2] + list4[4]` : non-numeric argument to binary operator
  - Error because: `[ ]` returns sublists (not the actual values). We're trying to add `list(42) + list(126)`

### 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.

```
?seq
```

```
seq(from = 1, to = 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
```

```
seq(from = 1, to = 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)`.

- `rep(1:3, times=3)`:
  - Repeats the **entire vector** 1:3 three times.
  - Output: 1 2 3 1 2 3 1 2 3
  - The sequence 1, 2, 3 is repeated back-to-back three times.
- `rep(1:3, each=3)`:
  - Repeats **each element** of the vector three times before moving to the next element.
  - Output: 1 1 1 2 2 2 3 3 3
  - Each number (1, then 2, then 3) is repeated three times consecutively.
- **Summary:**
  - `times` repeats the whole vector.
  - `each` repeats each element individually before moving to the next.

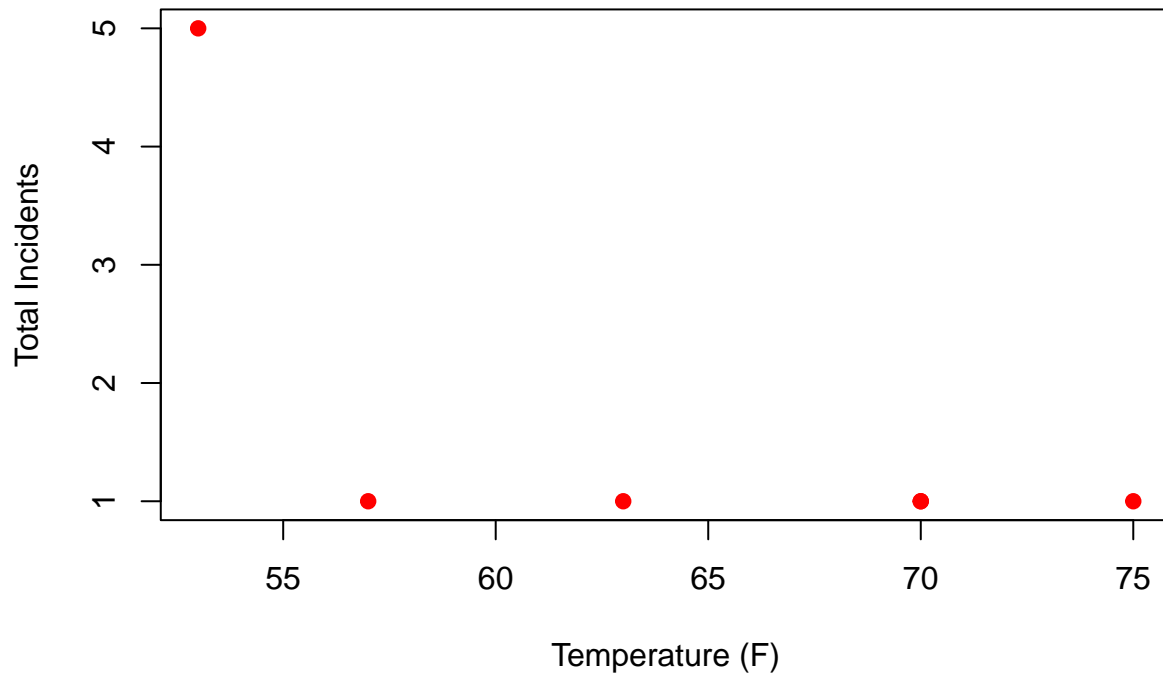
MB.Ch1.2. The `orings` data frame gives data on the damage that had occurred in US space shuttle launches prior to the disastrous Challenger launch of 28 January 1986. The observations in rows 1, 2, 4, 11, 13, and 18 were included in the pre-launch charts used in deciding whether to proceed with the 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.

```
# Create a new data frame by extracting these rows from orings
library(DAAG)
selected_rows <- c(1, 2, 4, 11, 13, 18)
orings_subset <- orings[selected_rows, ]

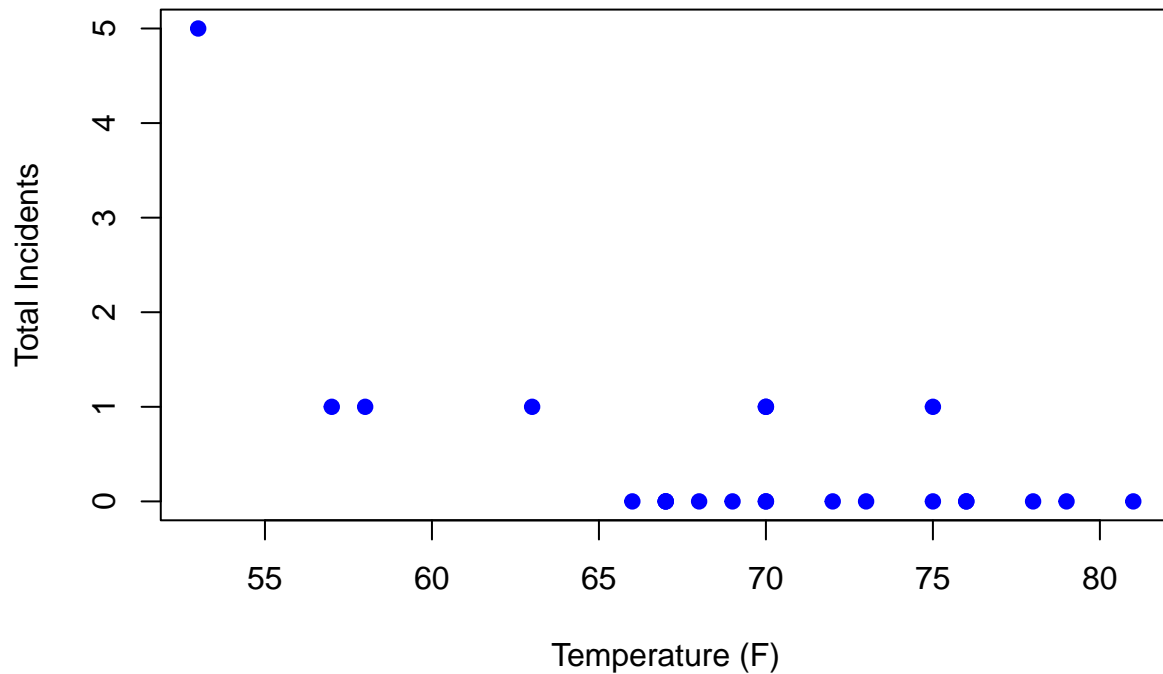
# Plot total incidents against temperature for this new data frame
plot(orings_subset$Temperature, orings_subset$Total,
     xlab = "Temperature (F)", ylab = "Total Incidents",
     main = "O-Ring Damage(New Data Frame)",
     pch = 19, col = "red")
```

## O-Ring Damage(New Data Frame)



```
# Obtain a similar plot for the full data set  
plot(orings$Temperature, orings$Total,  
      xlab = "Temperature (F)", ylab = "Total Incidents",  
      main = "O-Ring Damage (Full Data Set)",  
      pch = 19, col = "blue")
```

## O-Ring Damage (Full Data Set)



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.

```
# Load the DAAG package
library(DAAG)

# Load the ais dataset
data(ais)

# Get the structure of the ais data frame
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 ...

# Check for missing values in each column
colSums(is.na(ais))

##      rcc      wcc      hc      hg      ferr      bmi      ssf pcBfat      lbm      ht      wt
##       0       0       0       0       0       0       0       0       0       0       0
##      sex  sport
##       0       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?

```
# Load the data
library(DAAG)
data(ais)

# Create the table of counts
sex_by_sport <- table(ais$sport, ais$sex)
print(sex_by_sport)

##
##           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

# Find sports with >2:1 gender imbalance
imbalance <- sex_by_sport[, "f"] / sex_by_sport[, "m"] # female:male ratio

# Sports where females outnumber males by >2:1
female_dominant <- names(which(imbalance > 2))

# Sports where males outnumber females by >2:1
male_dominant <- names(which(imbalance < 0.5))

cat("\nSports with female > 2 male:", female_dominant)

##
## Sports with female > 2 male: Gym Netball

cat("\nSports with male > 2 female:", male_dominant)

##
## Sports with male > 2 female: T_Sprnt W_Polo
```

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

```
# Create the data frame
Manitoba.lakes <- data.frame(
  elevation = c(217, 254, 248, 254, 253, 227, 178, 207, 217),
  area = c(24387, 5374, 4624, 2247, 1353, 1223, 1151, 755, 657)
)

# Set row names (lake names)
row.names(Manitoba.lakes) <- c("Winnipeg", "Winnipegosis", "Manitoba",
  "SouthernIndian", "Cedar", "Island",
  "Gods", "Cross", "Playgreen")

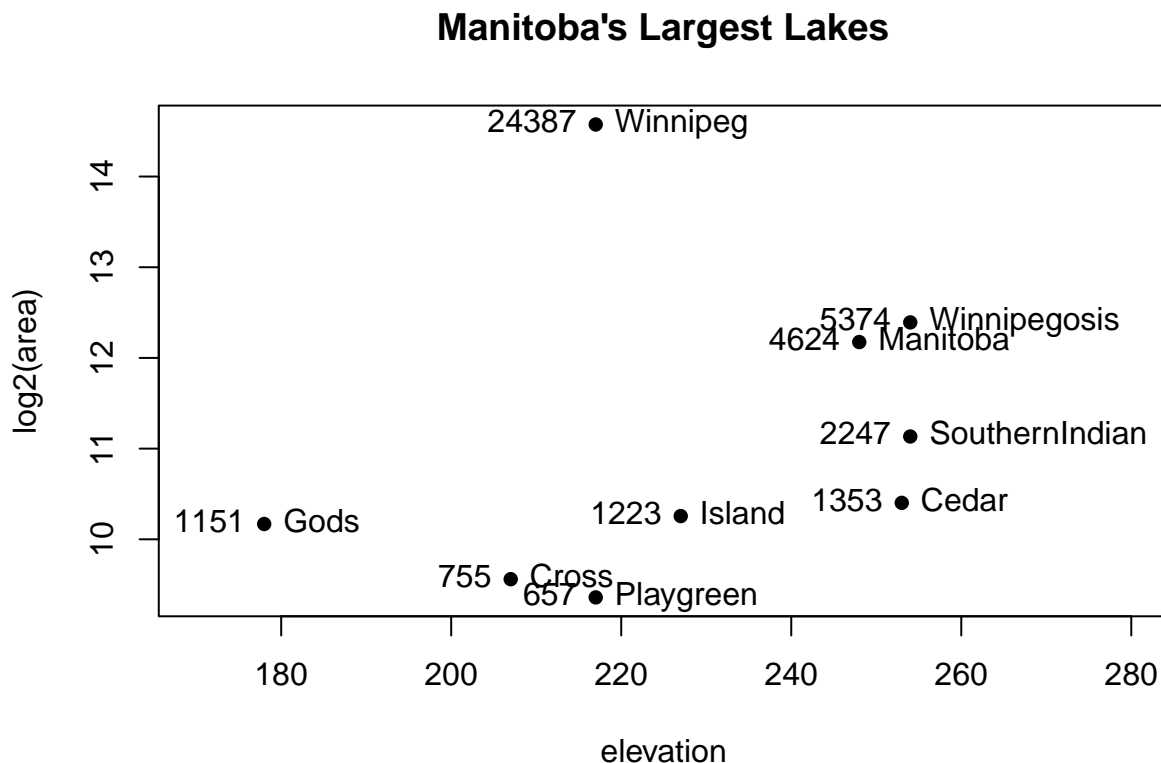
# Display the result
Manitoba.lakes

##           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
```

- (a) Use the following code to plot  $\log_2(\text{area})$  versus  $\text{elevation}$ , adding labeling information (there is an extreme value of area that makes a logarithmic scale pretty much essential):

```
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")
```

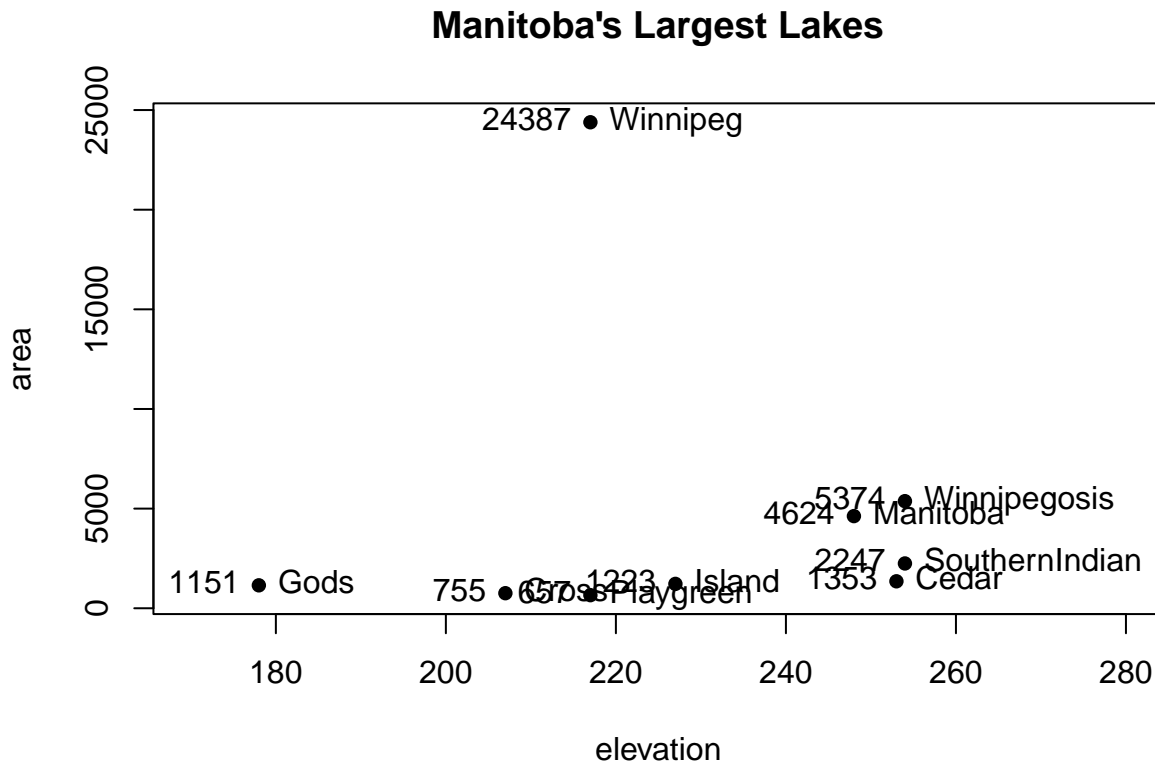


Devise captions that explain the labeling on the points and on the y-axis. It will be necessary to explain how distances on the scale relate to changes in area.

- Here are explanatory captions for the plot:
  - **Y-axis caption:** “The y-axis shows  $\log_2(\text{area})$ , where each unit increase represents a doubling of lake area. For example, Winnipeg ( $\log_2(\text{area}) \approx 14.6$ ) has approximately  $2^{(14.6-13.4)} \approx 2.3$  times the area of Winnipegosis ( $\log_2(\text{area}) \approx 13.4$ ).”
  - **Point labeling caption:** “Lake names are shown to the right of each point, while actual area values (in  $\text{km}^2$ ) appear to the left. The logarithmic transformation compresses the wide range of areas (657-24,387  $\text{km}^2$ ) into a manageable scale where Winnipeg appears as an extreme outlier.”
  - **Scale relationship explanation:** “Vertical distances on this plot correspond to ratios of areas. Moving up by 1 unit means an area is twice as large, while moving down by 1 unit means an area is half as large. The 15-unit range on the y-axis reflects that Winnipeg’s area is about  $2^{15} \sim 32,000$  times larger than the smallest lake’s area.”

- (b) Repeat the plot and associated labeling, now plotting area versus elevation, but specifying `ylog=TRUE` 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")
```

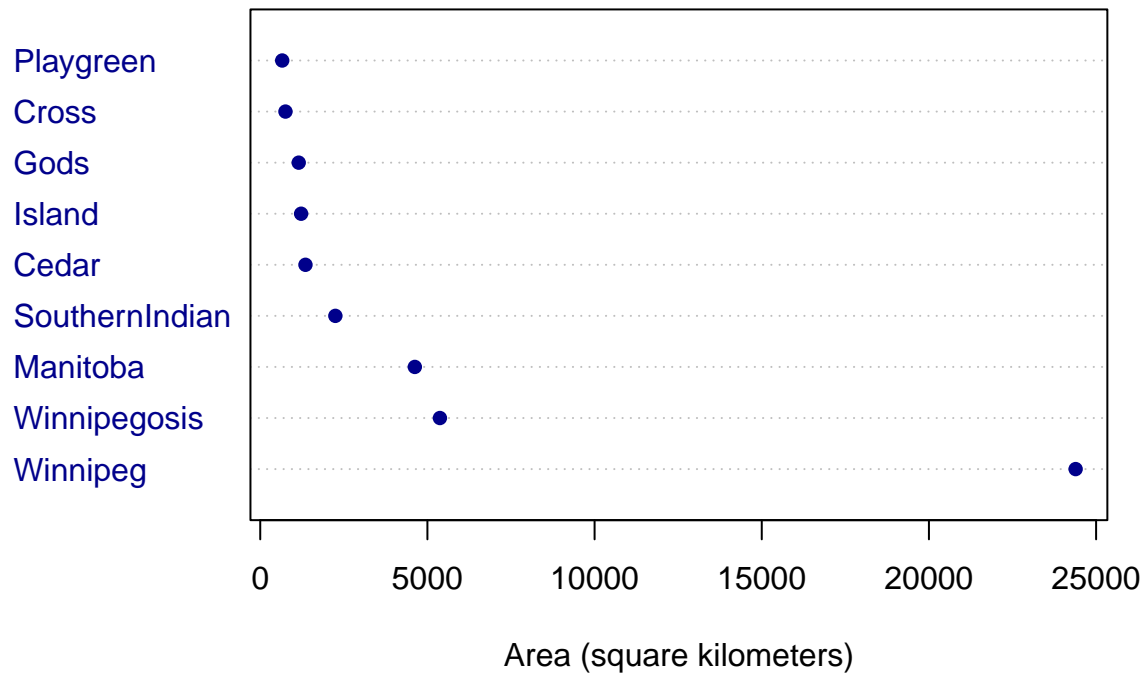


MB.Ch1.7. Look up the help page for the R function `dotchart()`. Use this function to display the areas of the Manitoba lakes (a) on a linear scale, and (b) on a logarithmic scale. Add, in each case, suitable labeling information.

```
# (a) Linear scale dotchart
dotchart(Manitoba.lakes$area,
  labels = rownames(Manitoba.lakes),
  main = "Surface Areas of Manitoba Lakes (Linear Scale)",
  xlab = "Area (square kilometers)",
  pch = 16,
  color = "darkblue")
```

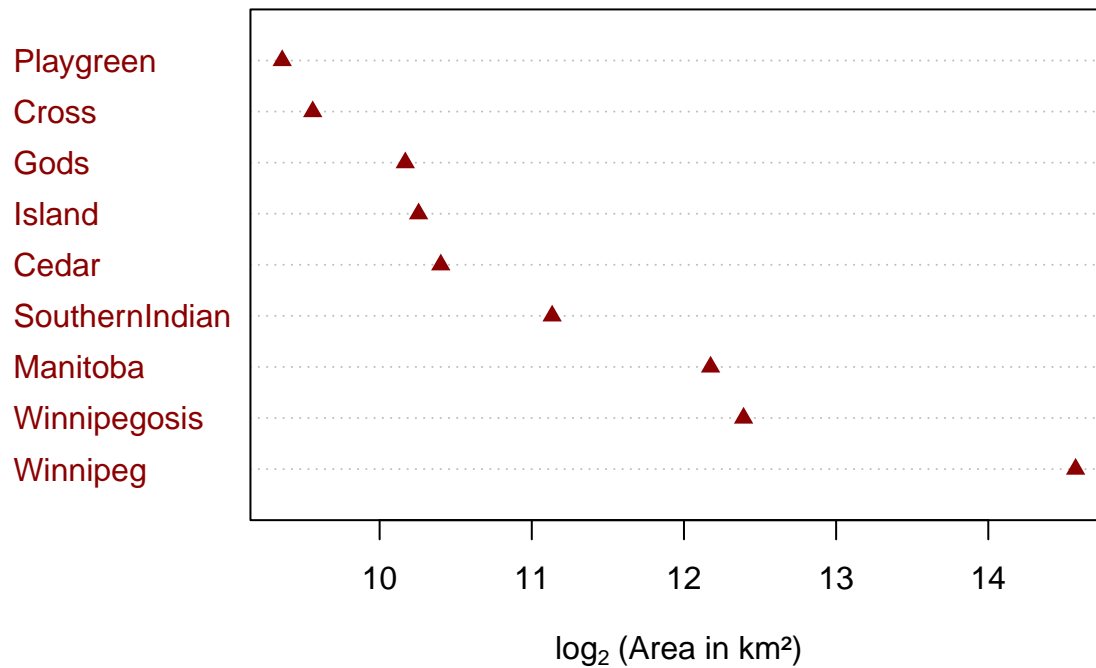


## Surface Areas of Manitoba Lakes (Linear Scale)



```
# (b) Logarithmic scale dotchart (preserving default axis)
dotchart(log2(Manitoba.lakes$area),
  labels = rownames(Manitoba.lakes),
  main = "Surface Areas of Manitoba Lakes (Logarithmic Scale)",
  xlab = expression(log[2]~"(Area in km²)"),
  pch = 17,
  color = "darkred")
```

## Surface Areas of Manitoba Lakes (Logarithmic Scale)



MB.Ch1.8. Using the `sum()` function, obtain a lower bound for the area of Manitoba covered by water.

```
# Calculate lower bound for water coverage
water_area_lower_bound <- sum(Manitoba.lakes$area)
```

```
# Print the result with proper formatting
cat("Lower bound for Manitoba's water coverage (major lakes only):",
    format(water_area_lower_bound, big.mark = ","),
    "square kilometers\n")
```

```
## Lower bound for Manitoba's water coverage (major lakes only): 41,771 square kilometers
```