Problem Set 2 - Solutions

Advanced Forecasting, UNCC, 2019

Ercan Karadas

Exercise

Store the values `-20, -15, -5, 8, 12, 9, 2, 23, 19` into the variable `x`.

```
x <- c(-20, -15, -5, 8, 12, 9, 2, 23, 19)
```

a. Use the R command `sum` to verify that the sum of the values is 33.

```
sum(x)
## [1] 33
```

b. Compute an average by using the `R` command `mean`?

```
mean(x)
## [1] 3.667
```

c. Compute the average with using the `R` command `sum`?

```
sum(x)/length(x)
## [1] 3.667
```

d. Use `R` to sum the positive values in `x`.

```
sum(x[x>0])
## [1] 73
```

e. Use the `which` command to get the average of the values ignoring the largest value.

```
mean(x[which(x!=max(x))])
## [1] 1.25
```

f. Speculate about the values corresponding to the command `x[abs(x)>=8 & x<8]`. Verify your speculation running this R command.

```
x[abs(x)>=8 & x<8]
## [1] -20 -15
```

Exercise

The final exam scores for 15 students are

$$73, 74, 92, 98, 100, 72, 74, 85, 76, 94, 89, 73, 76, 99$$

Compute the mean, 20% trimmed mean, and median using R.

```
scores <- c(73,74,92,98,100,72,74,85,76,94,89, 73, 76, 99)
mean(scores)
## [1] 83.93
mean(scores, trim=0.2)
## [1] 83.1
median(scores)
## [1] 80.5
```

Exercise

Let `x = c(1, 8, 2, 6, 3, 8, 5, 5, 5, 5)`

a. Describe two different R commands for summing the values in `x` ignoring the value 2 stored in `x[3]` and the value 3 stored in `x[5]`.

```
x <- c(1, 8, 2, 6, 3, 8, 5, 5, 5, 5)
#Method 1. There is an easy way of excluding a certain element of a vector.
#For example, the following command gets rid of the third element
x[-3]
## [1] 1 8 6 3 8 5 5 5 5
# So we could get rid of those two elements in two rounds :
x1 <- x[-5]
x2 <- x1[-3]
sum(x2)
## [1] 43

# Method 2
sum(x[c(1:2,4,6:length(x))])
## [1] 43
# Among all these probably the next one is best way to do it:
sum(x[-c(3,5)])
## [1] 43
```

b. Use two different R commands to sum all of the values not equal to 5.

```
# method 1
sum(x[x!=5])
## [1] 28

# method 2
sum(x[which(x!=5)])
## [1] 28
```

c. Use a single R command to change all values equal to 8 to 7.

```
x <- c(1, 8, 2, 6, 3, 8, 5, 5, 5, 5)
eights <- x==8  # or use eights <- which(x==8)
x[eights] <- 7
x
##  [1] 1 7 2 6 3 7 5 5 5 5

# of course we can combine these steps into one
x <- c(1, 8, 2, 6, 3, 8, 5, 5, 5, 5)
x[x==8] <- 7
x
##  [1] 1 7 2 6 3 7 5 5 5 5
```

Exercise

a. Create a 10×5 matrix M whose elements are random draws from a normal distribution with mean 5 and variance 2.

```
matrixData <- rnorm(50, mean=5, sd=2^2)
M <- matrix(matrixData, nrow=10, ncol=5)
```

```
M
##           [,1]    [,2]    [,3]    [,4]   [,5]
##  [1,]  5.2105  3.3629 11.1570  5.8532 9.3021
##  [2,] 19.9600  3.3093  0.3789 10.8564 4.5431
##  [3,] -0.9103  6.1791 -5.2169 12.4860 1.0147
##  [4,]  5.7923 -0.2374  1.4159  2.5801 6.9053
##  [5,]  1.6040  5.7570 -1.0096  8.9889 2.5288
##  [6,] 11.3173  1.2641  4.4132  0.9486 0.6741
##  [7,]  7.6630  5.6566  7.9408  1.3905 5.6474
##  [8,]  3.9491  3.4384 13.9335  0.7993 4.2983
##  [9,]  5.6125  4.5868  6.6105  5.7090 4.0720
## [10,]  4.0772  6.8605 10.7436  0.6635 1.8010
```

b. Create another matrix N of the same size which contains all zeros, except 5 NA and locations of these NAs are randomly determined (i.e. in the sense that each time you run your code their locations are expected to change).

```
N <- matrix(0, nrow=10, ncol=5)
naLocations <- sample(1:50, 5, replace=TRUE)
N[naLocations] <- NA
```

c. Using M and N generate a random matrix P from $N(5, 2)$ which contains 5 NA values that are arbitrarily located in the matrix.

```
P <- M + N
```

d. Describe how the R function `is.na` can be used to eliminate the rows with missing values.

```
naRows <- which(rowSums(is.na(P))>0)
naRows
## [1] 1 2 4 5 8
P[-naRows, ]
##          [,1]  [,2]   [,3]    [,4]   [,5]
## [1,] -0.9103 6.179 -5.217 12.4860 1.0147
## [2,] 11.3173 1.264  4.413  0.9486 0.6741
## [3,]  7.6630 5.657  7.941  1.3905 5.6474
## [4,]  5.6125 4.587  6.610  5.7090 4.0720
## [5,]  4.0772 6.860 10.744  0.6635 1.8010
```

Exercise

R has a built-in data set called `chickwts`, which is stored in a data frame with two columns. The first column contains the weight of chicks, and the second column indicates the type of feed they received, one of which is labeled `horsebean`. Use R to compute the average weight among chicks that were fed horsebean.

```
wantedIndices <- which(chickwts$feed == "horsebean")
mean(chickwts$weight[wantedIndices])
## [1] 160.2

# if you don't want to use which function
mean(chickwts$weight[chickwts$feed == "horsebean"])
## [1] 160.2
```

Exercise

a. Create a vector named `my_vec` containing the integers 1 through 100 and then divide each element of `my_vec` by 3 and store the result as `my_vec2`. (Your answer should contain two lines of R commands).

```
my_vec <- 1:100
my_vec2 <- my_vec/3
```

b. Compute the average of the vector `my_vec` you created before without using built-in R function `mean`.

```
mean(my_vec)/length(my_vec)
## [1] 0.505
```

c. Create a vector named `my_vec3` containing the elements of `my_vec` that are between 20 and 35. (Your answer should contain a single line of R commands)

```
my_vec3 <- my_vec[my_vec>=20 & my_vec<=35]
my_vec3
##  [1] 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
```

Exercise

Briefly explain what would be the output of following R command: `mean(rnorm(1000))`

```
This command first takes a sample of 1000 draws from N(0,1) and then computes the average
of that sample. This average should be very close to 0 because of the Law of Large Numbers.
```

Exercise

Make a script file which constructs two random normal vectors of length 10. Call these vectors `x1` and `x2`. Make a data frame called `T` with two columns (called `a` and `b`) containing respectively `x1` and `x1+x2`.

```
x1 <- rnorm(10)
x2 <- rnorm(10)
T <- data.frame(a=x1, b=x1+x2)
T
##           a       b
## 1  -1.15350 -1.0337
## 2  -0.22224 -1.5839
## 3   0.95947  1.3938
## 4  -0.26962 -0.2723
## 5  -0.64067 -0.6232
## 6  -0.07748 -1.8007
## 7  -0.30546  1.3626
## 8  -3.16537 -3.0536
## 9  -0.25220  0.6353
## 10  1.37860  2.2192
```

Exercise

When the function `head` called on the data frame `Orange` it produces the following output:

```
> head(Orange, n=2)
  Tree  age circumference
```

```
1    1  118            30
2    1  484            58
```

Write the command that adds up Y (defined as `circumference`) and X (defined as square root of `age`). (Your answer should contain at most three lines of R commands)

```
Y <- Orange$circumference
X <- sqrt(Orange$age)
X+Y
##  [1]  40.86  80.00 112.77 146.69 155.09 179.04 184.77  43.86  91.00 136.77
## [11] 187.69 207.09 240.04 242.77  40.86  73.00 100.77 139.69 150.09 176.04
## [21] 179.77  42.86  84.00 137.77 198.69 214.09 246.04 253.77  40.86  71.00
## [31] 106.77 156.69 177.09 211.04 216.77
```

Exercise (Dummy Variables)

`Orange` is a data frame with two numeric variables `circumference` and `age`. Create a dummy variable that is 0 if `age` is less than or equal to 900 and 1, otherwise.

```
Orange$age[Orange$age <= 900] <- 0
Orange$age[Orange$age > 900] <- 1
```

Exercise

Put all even integers from 30 to 89 in a vector named P and then in a matrix with 6 rows and 5 columns named Q. But make sure that the matrix is organized rowwise.

```
P <- seq(from=30, to=89, by=2)
Q <- matrix(P, nrow=6, ncol=5, byrow = TRUE)
Q
##      [,1] [,2] [,3] [,4] [,5]
## [1,]   30   32   34   36   38
## [2,]   40   42   44   46   48
## [3,]   50   52   54   56   58
## [4,]   60   62   64   66   68
## [5,]   70   72   74   76   78
## [6,]   80   82   84   86   88
```

Exercise

Declare a function in R, named `my_function` which takes a vector, say `x`, as input and returns the sum of the elements in the vector and the mean of the values in the vector. Also, make sure that this function stops and returns the message "Data is not numeric!" when you feed in a non-numeric vector.

```
my_function <- function(x){
  if (!is.numeric(x)) {stop("Data is not numeric")}
  mySum <- sum(x)
  myMean <- sum(x)/length(x)
  return(c(mySum, myMean))
}
my_function(x)
## [1] 46.0  4.6
```

Exercise

Using a `for` loop in R, write a script that produces the sum of the first $n = 40$ integers.

```
n = 50
sum <- 0
for(i in 1:n) {
sum <- sum + i }
sum
## [1] 1275

# We can put this into a user declared function, say mySum, as
mySum <- function(n){
sum <- 0
for(i in 1:n) {
sum <- sum + i }
return(sum)
}
# here n is the last integer. Let's try it out
mySum(5)
## [1] 15
```

Exercise

R has a built-in data set called `ChickWeight`. Verify that the R command

```
mean(ChickWeight[,1])
## [1] 121.8
```

returns 121.8 but that the command

```
mean(ChickWeight[,3])
## Warning in mean.default(ChickWeight[, 3]): argument is not numeric or
## logical: returning NA
## [1] NA
```

returns `NA` and a warning message even though the values in column 3 appear to be numeric. The reason for the warning message is that column 3 is stored as a factor variable. Arithmetic operations can only by performed on numeric or logical variables. Verify that

```
mean(as.numeric(ChickWeight[ ,3]))
## [1] 26.26
```

returns 26.26.

Exercise

Let $\{x_1, x_2, \ldots, x_n\}$ be a sample and suppose that we declare x_i an *outlier* if it is more than two standard deviation of the mean, i.e.

$$\frac{|x_i - \overline{X}|}{s} > 2\,.$$

For the values

$$-150, 30, 121, 132, 123, 145, 151, 119, 133, 134, 130, 200, 510$$

write an R command to determine whether any outliers exist.

```r
x <- c(-150, 30, 121, 132, 123, 145, 151, 119, 133, 134, 130, 200, 510)
myOutliers <- function(x){
  x_L = mean(x)-2*sd(x)
  x_U = mean(x)+2*sd(x)
  out <- x[x<x_L | x>x_U]
  return(out)
}
myOutliers(x)
## [1] -150  510
```

Exercise

Importing data from a plain text file can be illustrated with an example of a data set available on the website "Data and Story Library" (DASL). The Massachusetts lunatics data is available at http://lib.stat.cmu.edu/DASL/Datafiles/lunaticsdat.html.

a. Saved the data as `lunatics.txt` in a folder named "R Practice" on your desktop.

b. Use the `read.table` function to read the file into a data frame.

c. The `str` (structure) function provides a quick check that 14 observations of six variables.

d. Convert `lunatics.txt` into `lunatics.csv` and save in the same folder.

e. Import `lunatics.csv` using the package `readr` and compute the total population of 14 counties.

Exercise

Many of the interesting data sets that one may wish to analyze are available on a web page. R provides an easy way to access data from a file on the internet using the URL of the web page. The function `read.table` can be used to input data directly from the internet.

a. The data file `PiDigits.dat`, located at http://www.itl.nist.gov/div898/strd/univ/data/PiDigits.dat, contains the first 5000 digits of the mathematical constant π. Use `read.table` to save the data in the folder "R Practice".

b. Redo the same thing but now skip the first 50 digits and save as `PiDigitsSkipped.csv`.

c. Using `head` command print some of the data on the console.

d. Are the digits of π uniformly distributed? Using `table` command compute the relative frequencies of each digit.

e. Use `barplot` to visualize the tabulated data you obtained above.

Exercise

Let's you have two variables `A` and `B`

```r
A <- c(1:5)
B <- c(10:14)
```

a. Create a data frame `myData` with these two variables

```r
myData <- data.frame(A=c(1:5), B=c(10:14))
```

b. Create a new variable `mySum` that adds these two variables and a new variable called `myMean` that averages the two variables.

```
mySum <- myData$A + myData$B
myMean <- mySum/2
myData
##   A  B
## 1 1 10
## 2 2 11
## 3 3 12
## 4 4 13
## 5 5 14
```

c. Add these new variables to the original data frame. Here is how you do it in three different ways:

```
# Method 1: use `$` to append these two variables to the data frame directly
myData$mySum <- myData$A + myData$B
myData$myMean <- myMean
myData
##   A  B mySum myMean
## 1 1 10    11    5.5
## 2 2 11    13    6.5
## 3 3 12    15    7.5
## 4 4 13    17    8.5
## 5 5 14    19    9.5

# Method 2: use `attach()` function
attach(myData)
## The following objects are masked _by_ .GlobalEnv:
##
##     A, B, myMean, mySum
myData$mySum <- A + B
myData$myMean <- myMean
detach(myData)
myData
##   A  B mySum myMean
## 1 1 10    11    5.5
## 2 2 11    13    6.5
## 3 3 12    15    7.5
## 4 4 13    17    8.5
## 5 5 14    19    9.5

# Method 3: use `transform()` function
myData <- transform(myData,
                    mySum = A+B,
                    myMean = (A+B)/2)
myData
##   A  B mySum myMean
## 1 1 10    11    5.5
## 2 2 11    13    6.5
## 3 3 12    15    7.5
## 4 4 13    17    8.5
## 5 5 14    19    9.5
```

Exercise

Suppose you have the following data set

```
manager <- c(1, 2, 3, 4, 5)
date <- c("10/24/08", "10/28/08", "10/1/08", "10/12/08", "5/1/09")
country <- c("US", "US", "UK", "UK", "UK")
gender <- c("M", "F", "F", "M", "F")

age <- c(32, 45, 25, 39, 99)
        q1 <- c(5, 3, 3, 3, 2)
        q2 <- c(4, 5, 5, 3, 2)
        q3 <- c(5, 2, 5, 4, 1)
        q4 <- c(5, 5, 5, NA, 2)
        q5 <- c(5, 5, 2, NA, 1)

leadership <- data.frame(manager, date, country, gender, age,
                                  q1, q2, q3, q4, q5, stringsAsFactors=FALSE)
```

a. Recode the value 99 for `age` to indicate that the value is missing

b. Let's say you want to recode the ages of the managers in the leadership dataset from the continuous variable age to the categorical variable `agecat` (Young, Middle Aged, Elder).

c. Note that `agecat` is a character variable. Turn it into an ordered factor.

d. Let's say you want to change the variable `manager` to `managerID` and `age` to `ages`.

e. Suppose you have data `y <- c(1, 2, 3, NA)`. Then `is.na(y)` returns `c(FALSE, FALSE, FALSE, TRUE)`. What would be the output of the following command: `is.na(leadership[4:5, 6:10])`

f. Save the `date` variable as a date variable

Exercise

Explain why `sum(c(1, -2, NA, 3))` would return `NA`, but not `sum(c(1, -2, NA, 3), na.rm=TRUE)`?

Exercise

You can remove any observation with missing data by using the `na.omit()` function. It deletes any rows with missing data. Remove the row with NA values from `leadership` data frame.

Exercise

(Counting Missing Values) Create a vector of lenght 100 with every third element missing (namely `NA`) by only using vector operations. Then count the number of missing values by utilizing only the following two function `is.na` and `sum`.

Exercise

(Leadership Example continued) Suppose you have the following data

```
leadershipOriginal <- leadership
```

a. Create a new dataset `newdata` containing rows sorted from youngest manager to oldest manager.

b. Sorts the rows into female followed by male, and youngest to oldest within each gender.

c. Sorts the rows by gender, and then from oldest to youngest manager within each gender.

Exercise

Let's say you have a data frame named `mydata`, with variables `x1` and `x2`, and you want to create a new variable `sumx` that adds these two variables and a new variable called `meanx` that averages the two variables. If you use the code

```
sumx  <-  x1 + x2
meanx <- (x1 + x2)/2
```

What would you get?

You would get an error. The right way of doing it:

```
sumx  <-  mydata$x1 + mydata$x2
meanx <- (mydata$x1 + mydata$x2)/2
```

Exercise (sorting)

Suppose you have the following data

```
manager <- c(1, 2, 3, 4, 5)
date <- c("10/24/08", "10/28/08", "10/1/08", "10/12/08", "5/1/09")
country <- c("US", "US", "UK", "UK", "UK")
gender <- c("M", "F", "F", "M", "F")
age <- c(32, 45, 25, 39, 99)
q1 <- c(5, 3, 3, 3, 2)
q2 <- c(4, 5, 5, 3, 2)
q3 <- c(5, 2, 5, 4, 1)
q4 <- c(5, 5, 5, NA, 2)
q5 <- c(5, 5, 2, NA, 1)

leadershipOriginal <- data.frame(manager, date, country, gender, age,
                                 q1, q2, q3, q4, q5, stringsAsFactors=FALSE)
```

a. Create a new dataset `newdata` containing rows sorted from youngest manager to oldest manager.

```
leadership <- leadershipOriginal
   newdata <- leadership[order(leadership$age),]
   newdata
##   manager     date country gender age q1 q2 q3 q4 q5
## 3       3  10/1/08      UK      F  25  3  5  5  5  2
## 1       1 10/24/08      US      M  32  5  4  5  5  5
## 4       4 10/12/08      UK      M  39  3  3  4 NA NA
## 2       2 10/28/08      US      F  45  3  5  2  5  5
## 5       5   5/1/09      UK      F  99  2  2  1  2  1
```

b. Sort the rows into female followed by male, and youngest to oldest within each gender.

```
   newdata <- leadership[order(leadership$gender, leadership$age),]
   newdata
##   manager     date country gender age q1 q2 q3 q4 q5
```

```
## 3       3  10/1/08      UK      F  25  3  5  5  5  2
## 2       2 10/28/08      US      F  45  3  5  2  5  5
## 5       5   5/1/09      UK      F  99  2  2  1  2  1
## 1       1 10/24/08      US      M  32  5  4  5  5  5
## 4       4 10/12/08      UK      M  39  3  3  4 NA NA
```

c. Sort the rows by gender, and then from oldest to youngest manager within each gender.

```
   newdata <- leadership[order(leadership$gender, -leadership$age),]
   newdata
##   manager     date country gender age q1 q2 q3 q4 q5
## 5       5   5/1/09      UK      F  99  2  2  1  2  1
## 2       2 10/28/08      US      F  45  3  5  2  5  5
## 3       3  10/1/08      UK      F  25  3  5  5  5  2
## 4       4 10/12/08      UK      M  39  3  3  4 NA NA
## 1       1 10/24/08      US      M  32  5  4  5  5  5
```

Exercise (merging datasets)

Suppose we have these two datasets:

```
names_1 <- c("A", "B", "C")
names_2 <- c("A", "C", "B")
gender <- c("M", "F", "F")
age <- c(32, 45, 25)

dataFrame_1 <- data.frame(names_1, gender)
dataFrame_2 <- data.frame(names_2, age)
dataFrame_1
##   names_1 gender
## 1       A      M
## 2       B      F
## 3       C      F
dataFrame_2
##   names_2 age
## 1       A  32
## 2       C  45
## 3       B  25
```

a. Combine them in a meaningful way.

```
dataFrame_1new <- dataFrame_1[order(dataFrame_1$names_1),]
names(dataFrame_1new)[1] <- "names"

dataFrame_2new <- dataFrame_2[order(dataFrame_2$names_2),]
names(dataFrame_2new)[1] <- "names"

dataFrameOriginal <- merge(dataFrame_1new, dataFrame_2new, by = "names")
```

b. Add another variable, `income <- c(100, 85, 125)`, to the data frame by using `cbind`:

```
names <- c("A", "B", "C")
income <- c(100, 85, 125)

dataFrame_3 <- data.frame(names, income)
```

```
dataFrame <- cbind(dataFrameOriginal, dataFrame_3["income"])
dataFrame
##   names gender age income
## 1     A      M  32    100
## 2     B      F  25     85
## 3     C      F  45    125
```

c. Add another observation, `data.frame(names = c("D"), gender = c("F"), age = c(44))`, to the data frame by using `rbind`:

```
dataFrame_4 <- data.frame(names = c("D"), gender = c("F"), age = c(44))
dataFrame_4
##   names gender age
## 1     D      F  44

dataFrame <- rbind(dataFrameOriginal, dataFrame_4)
dataFrame
##   names gender age
## 1     A      M  32
## 2     B      F  25
## 3     C      F  45
## 4     D      F  44
```

d. Add another observation to the data frame by using `rbind` but this time two data frames have different colums, `data.frame(names = c("D"), gender = c("F"), age = c(44), occupation = c("teacher"))`:

```
dataFrame_5 <- data.frame(names = c("D"), gender = c("F"), age = c(44), occupation = c("teacher"))
dataFrame_5
##   names gender age occupation
## 1     D      F  44    teacher

# method 1: append the dataFrameOriginal
dataFrame <- cbind(dataFrameOriginal, occupation = c(NA))

dataFrame <- rbind(dataFrame, dataFrame_5)
dataFrame
##   names gender age occupation
## 1     A      M  32       <NA>
## 2     B      F  25       <NA>
## 3     C      F  45       <NA>
## 4     D      F  44    teacher
```

```
dataFrame_5 <- data.frame(names = c("D"), gender = c("F"), age = c(44), occupation = c("teacher"))
dataFrame_5
##   names gender age occupation
## 1     D      F  44    teacher

# method 2: delete the NA column
dataFrame <- rbind(dataFrameOriginal, dataFrame_5[1:3])
dataFrame
##   names gender age
## 1     A      M  32
## 2     B      F  25
## 3     C      F  45
```

```
## 4     D      F  44
```

Exercise (subsetting)

Selects variables q1, q2, q3, q4, and q5 from the leadership data frame and saves them to a dataframe newdata:

```
leadership <- leadershipOriginal
# method 1
newdata <- leadership[, c(6:10)]
newdata
##   q1 q2 q3 q4 q5
## 1  5  4  5  5  5
## 2  3  5  2  5  5
## 3  3  5  5  5  2
## 4  3  3  4 NA NA
## 5  2  2  1  2  1

# method 2
myvars <- c("q1", "q2", "q3", "q4", "q5")
newdata <- leadership[myvars]
newdata
##   q1 q2 q3 q4 q5
## 1  5  4  5  5  5
## 2  3  5  2  5  5
## 3  3  5  5  5  2
## 4  3  3  4 NA NA
## 5  2  2  1  2  1

# method 3: using paste() function
myvars <- paste("q", 1:5, sep = "")
newdata <- leadership[myvars]
newdata
##   q1 q2 q3 q4 q5
## 1  5  4  5  5  5
## 2  3  5  2  5  5
## 3  3  5  5  5  2
## 4  3  3  4 NA NA
## 5  2  2  1  2  1
```

Exercise (dropping variables)

Drop q3 and q4 from the leadership data frame and saves them to the data frame newdata.

```
leadership <- leadershipOriginal
# method 1
newdata <- leadership[, c(-8,-9)]
newdata
##   manager     date country gender age q1 q2 q5
## 1       1 10/24/08      US      M  32  5  4  5
## 2       2 10/28/08      US      F  45  3  5  5
## 3       3  10/1/08      UK      F  25  3  5  2
## 4       4 10/12/08      UK      M  39  3  3 NA
```

```
## 5       5   5/1/09      UK      F  99  2  2  1

# method 2
myvars <- names(leadership) %in% c("q3", "q4")
newdata <- leadership[!myvars]

# method 3
leadership$q3 <- leadership$q4 <- NULL
leadership
##   manager     date country gender age q1 q2 q5
## 1       1 10/24/08      US      M  32  5  4  5
## 2       2 10/28/08      US      F  45  3  5  5
## 3       3  10/1/08      UK      F  25  3  5  2
## 4       4 10/12/08      UK      M  39  3  3 NA
## 5       5   5/1/09      UK      F  99  2  2  1
```

Exercise (selecting observations)

Again use `leadership` data.

a. Select rows 1 through 3

```
leadership <- leadershipOriginal
newdata <- leadership[1:3,]
newdata
##   manager     date country gender age q1 q2 q3 q4 q5
## 1       1 10/24/08      US      M  32  5  4  5  5  5
## 2       2 10/28/08      US      F  45  3  5  2  5  5
## 3       3  10/1/08      UK      F  25  3  5  5  5  2
```

b. Select observations where men over 30

```
leadership <- leadershipOriginal
newdata <- leadership[leadership$gender == "M" & leadership$age > 30, ]
newdata
##   manager     date country gender age q1 q2 q3 q4 q5
## 1       1 10/24/08      US      M  32  5  4  5  5  5
## 4       4 10/12/08      UK      M  39  3  3  4 NA NA
```

c. Select observations where men over 30 and be careful with comma. Figure out what is going on below

```
leadership <- leadershipOriginal
newdata <- leadership[leadership$gender == "M" & leadership$age > 30]
newdata
##   manager gender q1 q4
## 1       1      M  5  5
## 2       2      F  3  5
## 3       3      F  3  5
## 4       4      M  3 NA
## 5       5      F  2  2
```

d. Limit your analyses to observations collected between January 1, 2009 and December 31, 2009.

```
leadership <- leadershipOriginal

# Converts the date values read in originally as character values to date values
```

```
# using the format mm/dd/yy
leadership$date <- as.Date(leadership$date, "%m/%d/%y")

# Create starting and ending dates
startdate <- as.Date("2009-01-01")
enddate <- as.Date("2009-10-31")

# Selects cases meeting your desired criteria
newdata <- leadership[which(leadership$date >= startdate &
           leadership$date <= enddate), ]

newdata
##   manager       date country gender age q1 q2 q3 q4 q5
## 5       5 2009-05-01      UK      F  99  2  2  1  2  1
```

Exercise (subset() function)

The `subset()` function is probably the easiest way to select variables and observations.

a. Selects all rows that have a value of age greater than or equal to 35 or less than 24. Keeps variables q1 through q4.

```
leadership <- leadershipOriginal

newdata <- subset(leadership, age >= 35 | age < 24,
                  select=c(q1, q2, q3, q4))
newdata
##   q1 q2 q3 q4
## 2  3  5  2  5
## 4  3  3  4 NA
## 5  2  2  1  2
```

b. Selects all men over the age of 25, and keeps variables gender through q4 (gender, q4, and all columns between them)

```
leadership <- leadershipOriginal

newdata <- subset(leadership, age > 25 & gender == "M",
                  select=c(gender:q4))
newdata
##   gender age q1 q2 q3 q4
## 1      M  32  5  4  5  5
## 4      M  39  3  3  4 NA
```

Exercise (sample() function)

The `sample()` function enables you to take a random sample (with or without replacement) of size n from a dataset.

Take a random sample of size 3 from the leadership dataset

```
leadership <- leadershipOriginal
mysample <- leadership[sample(1:nrow(leadership), 3, replace=FALSE),]
mysample
##   manager     date country gender age q1 q2 q3 q4 q5
```

```
## 1       1 10/24/08      US      M  32  5  4  5  5  5
## 5       5   5/1/09      UK      F  99  2  2  1  2  1
## 2       2 10/28/08      US      F  45  3  5  2  5  5
```