R programming practice

Khanin

2/11/2022

1.

1a. Create a matrix:

```
m <- matrix(c(1:10, 11:20), nrow = 10, ncol = 2)
m
```

```
##       [,1] [,2]
##  [1,]    1   11
##  [2,]    2   12
##  [3,]    3   13
##  [4,]    4   14
##  [5,]    5   15
##  [6,]    6   16
##  [7,]    7   17
##  [8,]    8   18
##  [9,]    9   19
## [10,]   10   20
```

Now find the mean of the rows

```
apply(m, 1, mean)
```

```
##  [1]  6  7  8  9 10 11 12 13 14 15
```

And find the mean of the columns

```
apply(m, 2, mean)
```

```
## [1]  5.5 15.5
```

Divide all values by 2

```
apply(m, c(1, 2), function(x) x/2)
```

```
##       [,1] [,2]
##  [1,]  0.5  5.5
##  [2,]  1.0  6.0
##  [3,]  1.5  6.5
##  [4,]  2.0  7.0
##  [5,]  2.5  7.5
##  [6,]  3.0  8.0
##  [7,]  3.5  8.5
##  [8,]  4.0  9.0
##  [9,]  4.5  9.5
## [10,]  5.0 10.0
```

1b. Examine the following R code. Explain (in English, not in code) what it is doing.

```r
z.sq <- function(z) return(c(z,z^2))
```

This z.sq function receives an input z and returns a vector of z and z^2.

```r
x <- 1:8
```

Create a vector that has values from 1 to 8

```r
z.sq(x)
```

```
##  [1]  1  2  3  4  5  6  7  8  1  4  9 16 25 36 49 64
```

Pass x into the z.sq function, returning both x and x^2 for each element as a vector

```r
matrix(z.sq(x),ncol=2)
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    2    4
## [3,]    3    9
## [4,]    4   16
## [5,]    5   25
## [6,]    6   36
## [7,]    7   49
## [8,]    8   64
```

Convert the x vector into a matrix that has two columns and fill values by columns
How could you simplify this?

```r
matrix(as.vector(sapply(x, function(z) return(c(z,z^2)))), ncol=2, byrow = TRUE)
```

```
##      [,1] [,2]
## [1,]    1    1
## [2,]    2    4
## [3,]    3    9
## [4,]    4   16
## [5,]    5   25
## [6,]    6   36
## [7,]    7   49
## [8,]    8   64
```

I could simplify this by using sapply and orphanage function, then convert the output to a matrix that fills values by rows.

2. Suppose we have a matrix of 1s and 0s, and we want to create a vector that has a 1 or a 0 depending on whether the majority of the first c elements in that row are 1 or 0. Here c will be a parameter which we can vary. Write a short function, perhaps called find.majority, that does this. Then apply it to the following matrix X when c=2 and c=3:

```
find.majority = function(z, c){
  ## Select only the top c variables
  subset_z = z[1:c]

  ## Count a total of 1 and 0 from subset_z for a given row
  length.1 = length(which(subset_z==1))
  length.0 = length(which(subset_z==0))

  if (length.1 > length.0) {  # If length(1) > length(0), it returns the length of 1.
    return(c(1, length.1))
  }
  else if (length.0 > length.1) { # If length(0) > length(1), it returns the length of 0.
    return(c(0, length.0))
  }
  else {
    # If length(0) = length(1), it returns "Same length" because the it
    # truly represents the result.
    return(c("Same length", length.1))
  }
}
```

Define a matrix X

```
X <- matrix(c(1,1,1,0, 0,1,0,1, 1,1,0,1, 1,1,1,1, 0,0,1,0), nrow=4)
```

For c = 2

```
c = 2
apply(X, 1, find.majority, c) ## The apply(X, 1,...) means iteration over each row of X.
```

```
##      [,1]          [,2] [,3]          [,4]
## [1,] "Same length" "1"  "Same length" "Same length"
## [2,] "1"           "2"  "1"           "1"
```

For c = 3

```
c = 3
apply(X, 1, find.majority, c)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    1    0    1
## [2,]    2    3    2    2
```

To interpret the results of these two c, the first row of the returned matrix indicates the majority of 1, 0, or same length when the number of 1 is equal to 0. The second row shows the its occurrence of the result in the first row. "Same length" means the number of 1 and 0 given the c is equal.
NOTE: since I return the result as "Same length", those who need to further use this function need to convert the values of the second row to be integer as the function would return string when the number of 1 is equal to 0.

3. EDA for Iris dataset

```
data(iris) ## Load iris
dim(iris)  ## Show dimension of iris
```

```
## [1] 150   5
```

```
head(iris) ## Show the top 6 rows of iris
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
class(iris) ## Show the class
```

```
## [1] "data.frame"
```

Find the mean petal length by species

```
tapply(iris$Petal.Length,iris$Species,mean)
```

```
##     setosa versicolor  virginica
##      1.462      4.260      5.552
```

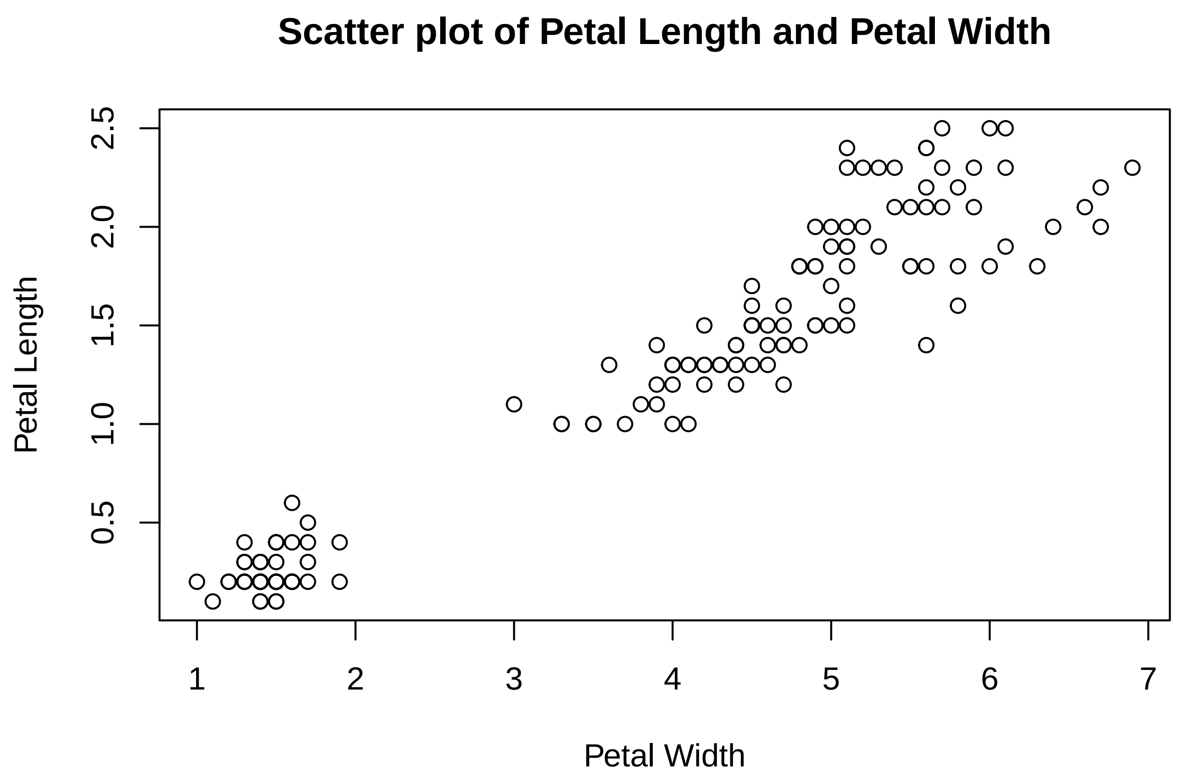
Now obtain the mean of the first 4 variables, by species, but using only one function call

```r
mean.first_4_variables_by_species = aggregate(cbind(iris$Sepal.Length, iris$Sepal.Width,
                                                    iris$Petal.Length, iris$Petal.Width),
                                                    list(iris$Species), FUN = mean)
## Assign column names to be readable
colnames(mean.first_4_variables_by_species) <- c('Species','Sepal.Length','Sepal.Width',
                                                 'Petal.Length', 'Petal.Width')
mean.first_4_variables_by_species
```

```
##      Species Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1     setosa        5.006       3.428        1.462       0.246
## 2 versicolor        5.936       2.770        4.260       1.326
## 3  virginica        6.588       2.974        5.552       2.026
```

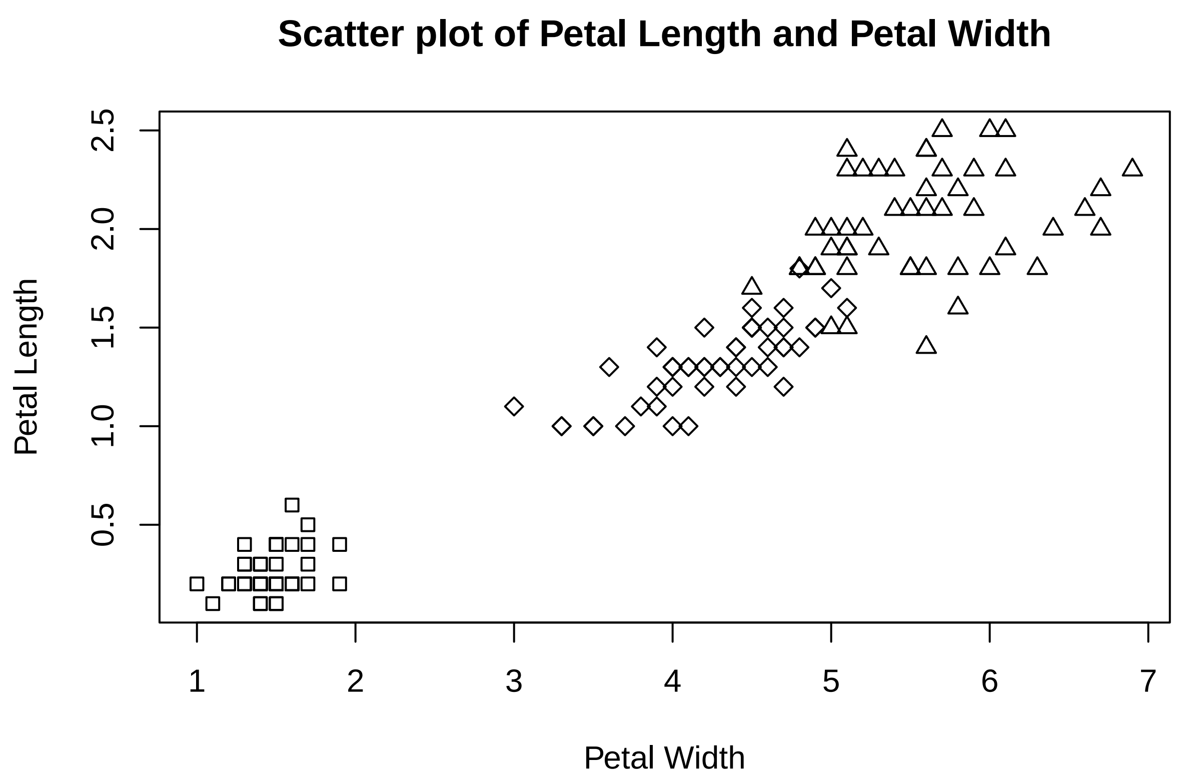
Create a simple scatter plot of Petal Length against Petal Width. Title your plot.

```r
plot(iris$Petal.Length, iris$Petal.Width, main='Scatter plot of Petal Length and Petal Width',
     xlab = 'Petal Width', ylab = 'Petal Length')
```



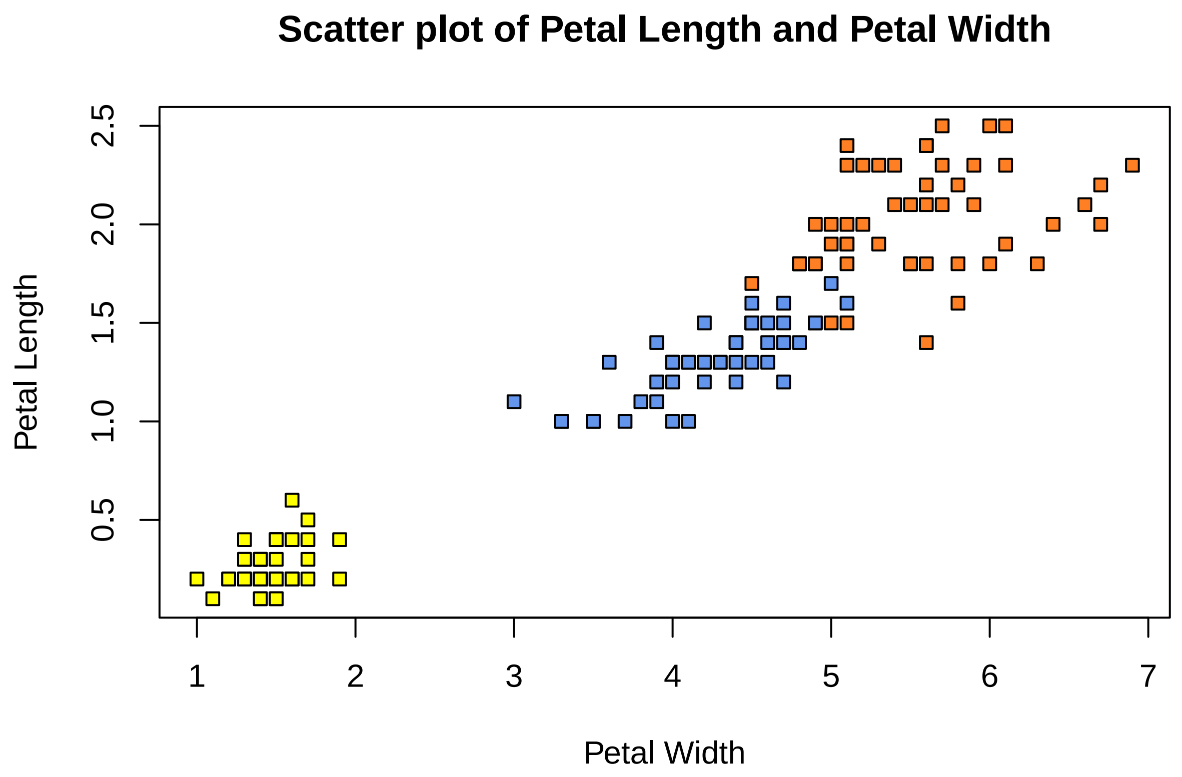
Now change the plotting symbol to be different for each species

```r
plot(iris$Petal.Length, iris$Petal.Width, pch=c(22,23,24)[unclass(iris$Species)],
     main='Scatter plot of Petal Length and Petal Width', xlab = 'Petal Width',
     ylab = 'Petal Length')
```



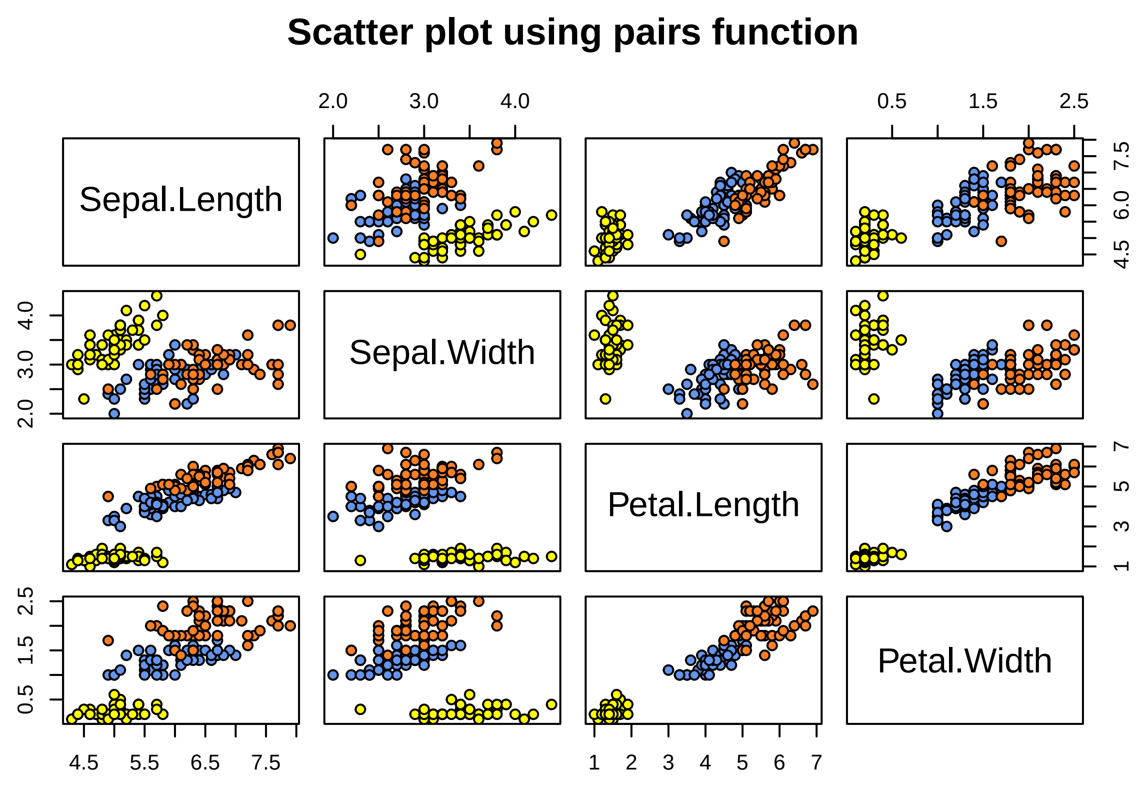
Replot the data using the same symbol for all species, but plot each species in a differet color, filling the symbol

```
plot(iris$Petal.Length, iris$Petal.Width, pch=c(22),
     bg=c("yellow", "cornflowerblue", "chocolate1")[unclass(iris$Species)],
     main='Scatter plot of Petal Length and Petal Width', xlab = 'Petal Width',
     ylab = 'Petal Length')
```



A very useful function in R is "pairs." Use the pairs function to create a plot of the iris data, comparing Petal Length, Petal Width, Sepal Length, and Sepal Width. You should have 12 subplots. Use the previous question to code each of the points in a different color by species.

```
pairs(iris[1:4], main='Scatter plot using pairs function',
      pch = 21, bg = c("yellow", "cornflowerblue",
                       "chocolate1")[unclass(iris$Species)])
```



What can you conclude about the data, on inspection of the pairs plot?

With the objective to predict the species of iris dataset, these features seem to be able to predict the categories of species because of the clear distinction of the color areas. However, sepal length and sepal width should not be used as main features alone since they would make it difficult to differentiate versicolor (blue) and virginica (orange).

4. Apply functions. Create a list with 2 elements:

```
l <- list(a = 1:10, b = 11:20)
l
```

```
## $a
##  [1]  1  2  3  4  5  6  7  8  9 10
##
## $b
##  [1] 11 12 13 14 15 16 17 18 19 20
```

What is the mean of the values in each element?

```
lapply(l, mean)
```

```
## $a
## [1] 5.5
##
## $b
## [1] 15.5
```

What is the sum of the values in each element?

```
lapply(l, sum)
```

```
## $a
## [1] 55
##
## $b
## [1] 155
```

What type of object is returned if you use lapply? sapply?

```
lapply(l, mean) # returns a list
```

```
## $a
## [1] 5.5
##
## $b
## [1] 15.5
```

```
sapply(l, mean) # returns a vector
```

```
##    a    b
##  5.5 15.5
```

lapply returns a list, and sapply returns a vector.

Now create the following list:

```
l.2 <- list(c = c(21:30), d = c(31:40))
l.2
```

```
## $c
##  [1] 21 22 23 24 25 26 27 28 29 30
##
## $d
##  [1] 31 32 33 34 35 36 37 38 39 40
```

What is the sum of the corresponding elements of l and l.2?

```
mapply("+", l, l.2, SIMPLIFY = FALSE)
```

```
## $a
##  [1] 22 24 26 28 30 32 34 36 38 40
##
## $b
##  [1] 42 44 46 48 50 52 54 56 58 60
```

Take the log of each element in the list l:

```
lapply(l, log)
```

```
## $a
##  [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
##  [8] 2.0794415 2.1972246 2.3025851
##
## $b
##  [1] 2.397895 2.484907 2.564949 2.639057 2.708050 2.772589 2.833213 2.890372
##  [9] 2.944439 2.995732
```

log2 of each value in each list

```
lapply(l, log2)
```

```
## $a
##  [1] 0.000000 1.000000 1.584963 2.000000 2.321928 2.584963 2.807355 3.000000
##  [9] 3.169925 3.321928
##
## $b
##  [1] 3.459432 3.584963 3.700440 3.807355 3.906891 4.000000 4.087463 4.169925
##  [9] 4.247928 4.321928
```

```
lapply(l.2, log2)
```

```
## $c
##  [1] 4.392317 4.459432 4.523562 4.584963 4.643856 4.700440 4.754888 4.807355
##  [9] 4.857981 4.906891
##
## $d
##  [1] 4.954196 5.000000 5.044394 5.087463 5.129283 5.169925 5.209453 5.247928
##  [9] 5.285402 5.321928
```

5. Write a function that finds the sample covariance:

This is a function to find the sample covariance.
Input: Dataset mat
Output: Covariance Matrix
### **Covariance formula:** link

```r
sampcov <- function( mat ) {

  # find the mean for each column, called the sample.mean
  sample.mean = sapply(mat,FUN=mean)

  # subtract the sample mean from each observation
  sample.subtract = mapply("-", mat, sample.mean)

  # implement matrix multiplication (hint: leave the following code as it is)
  yyt <- function(y) return(t(y) %*% y)

  # now use apply() to carry out matrix multiplication over the rows of Mat
  # notice the output will have ncol(Mat)^2 rows, nrow(Mat) columns

  return(apply(yyt(sample.subtract), c(1, 2), function(x) x/(dim(mat)[1]-1)))
}
```

Call the sampcov function to get the result

```r
sampcov(iris[,1:4])
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    0.6856935  -0.0424340    1.2743154   0.5162707
## Sepal.Width    -0.0424340   0.1899794   -0.3296564  -0.1216394
## Petal.Length    1.2743154  -0.3296564    3.1162779   1.2956094
## Petal.Width     0.5162707  -0.1216394    1.2956094   0.5810063
```

Compare the result with the cov (built-in) function.

```r
cov(iris[,1:4])
```

```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    0.6856935  -0.0424340    1.2743154   0.5162707
## Sepal.Width    -0.0424340   0.1899794   -0.3296564  -0.1216394
## Petal.Length    1.2743154  -0.3296564    3.1162779   1.2956094
## Petal.Width     0.5162707  -0.1216394    1.2956094   0.5810063
```