STATS 406 Fall 2016: Lab 06

1 R

1.1 Vectors

A vector contains elements of the same type, such as integer, numeric, character, and others. There are various ways to get a subset of a vector. And there are many operations that can be carried out on a vector. Many methods on vectors are the basis for more complicated data structures.

```
v = c(3,1,5,9,7)
print(v)

## [1] 3 1 5 9 7

length(v)

## [1] 5

# access elements by []
v[1]

## [1] 3

v[1:3]

## [1] 3 1 5

v[-1]

## [1] 1 5 9 7

# use logic values to select subset
v[v>4]

## [1] 5 9 7

# get the index satisfying certain conditions
which(v>4)

## [1] 3 4 5

# maximum of a vector
max(v)
```

```
## [1] 9

# index that maximizes the vector
which.max(v)

## [1] 4

# average of a vector
mean(v)

## [1] 5

# calculation on logical vectors
sum(v>4)

## [1] 3

# sort increasingly
sort(v)

## [1] 1 3 5 7 9

# sort decreasingly
sort(v, decreasing = T)

## [1] 9 7 5 3 1

# order of elements increasingly
order(v)

## [1] 2 1 3 5 4

# order of elements decreasingly
order(v,decreasing = T)

## [1] 4 5 3 1 2
```

1.2 Lists

A list can store elements that are not necessarily of the same type. Subsetting lists is different from subsetting a vector.

```r
list1 = list(c(2,5,3),21.3,"hello")
print(list1)
```

```
## [[1]]
## [1] 2 5 3
##
## [[2]]
## [1] 21.3
##
## [[3]]
## [1] "hello"
```

```r
# use [[]] to subset lists
list1[[1]]
```

```
## [1] 2 5 3
```

```r
list2 = list(c(1,2,3,4),c(5,6,7),c(8,9))
print(list2)
```

```
## [[1]]
## [1] 1 2 3 4
##
## [[2]]
## [1] 5 6 7
##
## [[3]]
## [1] 8 9
```

```r
# use lapply and sapply to run a function on every element of the list
lapply(list2,sum)
```

```
## [[1]]
## [1] 10
##
## [[2]]
## [1] 18
##
## [[3]]
## [1] 17
```

```r
sapply(list2,sum)
```

```
## [1] 10 18 17
```

1.3 Matrices

A matrix is two-dimensional and stores elements of the same type. There are various ways to subset a matrix and many operations for matrices.

```
A = matrix(1:6,nrow=3,ncol=2)
# In R, the default is by column
print(A)

##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6

B = matrix(1:6,nrow=3,ncol=2,byrow=T)
print(B)

##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6

# subset of a matrix
A[1,2]

## [1] 4

A[2,]

## [1] 2 5

A[,1]

## [1] 1 2 3

# elementwise operations
A+B

##      [,1] [,2]
## [1,]    2    6
## [2,]    5    9
## [3,]    8   12

A*B
```

```
##      [,1] [,2]
## [1,]    1    8
## [2,]    6   20
## [3,]   15   36

# matrix multiplication
A%*%t(B)

##      [,1] [,2] [,3]
## [1,]    9   19   29
## [2,]   12   26   40
## [3,]   15   33   51

# apply a function to every row/column
apply(A,1,max)

## [1] 4 5 6

apply(A,2,max)

## [1] 3 6
```

1.4 Dataframes

Dataframes are two-dimensional and they allow columns to be of different types. There are some similarities dealing with dataframes comparing to matrices. The following data file "student-mat.csv" is from Lab 2.

```
student = read.table("student-mat.csv",header=T,sep=";")
# subset of a dataframe
student[1,3]
student$age[1]
student[1,]
# select subset satisfying certain conditions
tmp = student[student$age==18,]
# operations
sum(student$age==18)
which(student$age>18)
# average of three grades G1, G2, G3
avg_G = apply(student[,c("G1","G2","G3")],1,mean)
```

1.5 For loops

For loops can be used to carry out a sequence of operations sharing some common patterns. Sometimes (not always) we can express a for loop in more efficient ways without the loop.

```
s = 0
for(i in 1:100){
  s = s + i
}
print(s)

## [1] 5050

# another way without using for loop
sum(1:100)

## [1] 5050
```

1.6 While loops

When the number of iterations is not known, we can use while loops to do a sequence of operations. The code below calculates the largest K such that $\sum_{i=1}^{K} i \leq 200$.

```
s = 0
i = 0
while(s<=200){
  i = i + 1
  s = s + i
}
K = i - 1
print(K)

## [1] 19
```

1.7 Functions

We can call and use existing functions in R. Also we can define functions by ourselves.

```
f <- function(n){
  return(sum(1:n))
}
f(100)

## [1] 5050
```

2 SQL

We can use the language SQL to work with relational databases.

The basic form of "SELECT" in SQL is:

```
sqltext1="
SELECT ColumnNames
FROM TableName
WHERE Conditions
GROUP BY VariableNames HAVING Conditions
ORDER BY VariableNames;"
```

We can also join two tables to get data, with the following basic form:

```
sqltext2="
SELECT T1.ColumnNames, T2.ColumnNames
FROM T1 INNER JOIN T2 ON JoiningConditions;"
```

An example from Lab 4: Using "baseball.db", join table *Schools* with table *SchoolsPlayers*. Generate a table showing the number of players from each school in Michigan. Sort by the number of players in descending order.

```
library("RSQLite")
driver = dbDriver("SQLite")
conn = dbConnect(driver, "baseball.db")
sqltext="SELECT Schools.schoolID, schoolName, COUNT(playerID) AS NumOfPlayers
                FROM Schools INNER JOIN SchoolsPlayers
                ON Schools.schoolID = SchoolsPlayers.schoolID
                WHERE schoolState = 'MI'
                GROUP BY Schools.schoolID
                ORDER BY NumOfPlayers DESC;"

dt = dbGetQuery(conn,sqltext)
head(dt)

##     schoolID                  schoolName NumOfPlayers
## 1   michigan      University of Michigan           72
## 2 michiganst   Michigan State University           37
## 3  wmichigan Western Michigan University           31
## 4  emichigan Eastern Michigan University           14
## 5  cmichigan Central Michigan University           13
## 6    detroit University of Detroit Mercy            8
```

3 XML

XML can be used to write documents that store both the data and data descriptions. It has a tree-like format. Note that XML is case-sensitive. An example from Lab 5:

```
<?xml version="1.0" encoding="UTF-8"?>
<students>
  <person SocialID="123" SchoolID="92031482">
    <name>Jim Brown</name>
    <gender>Male</gender>
    <major>Mathematics</major>
    <minor> Statistics </minor>
  </person>
  <person SocialID="234" SchoolID="91405720">
    <name>Susan Leicester</name>
    <gender>Female</gender>
    <major>Chemistry</major>
  </person>
</students>
```

Here “students” is the **root**, which is the first **node**. Its first child **node** has **name** “person”. This node has two **attributes** “SocialID” and “SchoolID”.