Session 2: Data Manipulation in R

Objectives:

By the end of this session, participants will be able to perform following tasks:

- Reading Comma separated files in R
- Extracting parts of data using **Indexing**
- Selecting specific columns and rows using **subset** command
- Generating custom summaries of data using **aggregate** and **tapply**
- Viewing help file of a command

In this session, we will be understanding how to manipulate data in R. The first step is to read data available in csv file. Using the command below, we will first load the breast cancer related data into a variable called *records.*

```
records <- read.csv("breastcancer.csv")
```

The next step is to understand the structure of file i.e., columns and rows present in the data using the following set of commands.

```
names(records)

dim(records)

head(records)

str(records)
```

2.1 Indexing

Often it happens when you want to work with parts of your dataset. We want to subset the data by selecting specific columns (variables) or particular rows. Both data frames and matrices have implicit row and column numbers that allow you to subset parts of your data. The process is called indexing. We can use this indexing to grab any portion of a data frame or matrix.

The command below will return the first column of our data frame *records.*

```
records[,1]
```

The command below will return the second column of our data frame *records.*

```
records[,2] #returns BMI of patients
```

To get the first row of the data frame, type the following command.

```
records[1,]
```

We can select more than one column and rows. The command below will return first three columns.

```
records[,1:3]
```

Type the following command to get first three rows of first three columns.

```
records[1:3,1:3]
```

We can also refer to columns by their names. For example, the command below will returns the rows of Classification column.

```
records[,"Classification"] #1 for healthy, 2 for Breast cancer patients
```

Another way to isolate a column in a data frame using the following syntax employing the symbol **$:**

```
records$Age
```

Exercise 2.1

What will be the output of records[1:4,] and records[1:5,1:5]?

2.2 Subset()

We can also use **subset** command to grab parts of the data. It is powerful, logical, and allows you to subset your data by specific columns and values in those columns. Below is the syntax of subset function.

```
subset(data.frame.name, column.name == value)
```

The command below will return the patients suffering from cancer in the dataset.

```
cancer_patients <- subset(records,records$Classification == 2)

cancer_patients
```

Exercise 2.2

Type the command to get the patients with age greater than 50?

In R, we use "&" for AND, "|" (the pipe) for OR, "!" for not, and "!=" for not equal. We can subset the data based on more than one condition. The command below will return the patients with age above 20 years and less than 50 years.

```
age_above20andbelow50 <- subset(records,records$Age>20 & records$Age<50)
```

2.3 Aggregate

In R, to get help of any function we can type the command name following the '**?**'**(question mark)** symbol. Let's first open the help file of aggregate function using the following command.

```
?aggregate
```

Aggregate function helps in generating custom summaries of data. For example, the command below will compute the average age of healthy individuals and cancer patients.

```
aggregate(records$Age, by = list(records$Classification), mean)
```

Use the following command to get the size of healthy individuals and cancer patients.

```
aggregate(records$Age, by = list(records$Classification), length)
```

Exercise 2.3

Compute the average BMI of healthy indiduals and cancer patients.

2.4 tapply()

tapply returns the same values that aggregate does, using a very similar structure to the arguments in aggregate but, instead of returning a data frame, it returns a matrix.

Exercise 2.4

Type the command to get the help of tapply function.

In the code below, tapply() function takes the values from the "Age" column (records$Age) and splits them into groups based on the values from the "Classification" column (list(records$Classification)). Then, it applies the mean() function to each group separately, calculating the mean age for each group.

```
tapply(records$Age,list(records$Classification), mean)
```

The output of this code will be a matrix containing average of Age for each group in the "Classification" column.

Exercise 2.5

Compute the average of Glucose for each classification group using tapply function.