**Data Analytics (CMP330)**

# Practical 2 – R Programming

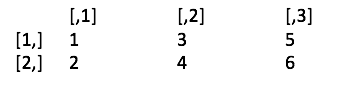
1. **Matrices**

Matrices are 2 dimensional vectors (or 2D arrays)

**m <- matrix( c(1,2,3,4,5,6), nrow = 2, ncol = 3)**

**m**

Should print out:



Try the following code, line by line:

**dim(m)**

**nrow(m)**

**ncol(m)**

**colnames(m) <- c('a','b','c')**

**m[,'b']**

Try to work out what the following statements will do before running them:

**m[2,2]**

**m[,2]**

**m[1,]**

**m[,1:2]**

1. **The apply() function**

Read about the apply function by typing in

**?apply**

The structure of the apply function is as follows:apply(m, dimcode, f, fargs)

Now try the apply function with the mean function:

**x <- matrix(1:20, nrow = 10, ncol = 2)**

**apply(x, 2, mean)**

Now try the following code and try to understand each statement:

**x <- matrix(1:20, nrow = 10, ncol = 2)**

**rowSums <- apply(x, 1, sum)**

**rowMeans <- apply(x, 1, mean)**

**colSums <- apply(x, 2, sum)**

**colMeans <- apply(x, 2, mean)**

1. **Factors**

Factors represent nominal/categorical data.

Try the following code:

**?factor**

**x <- factor( c("yes", "yes", "no", "yes", "no") )**

**x**

The print-out should be:

**[1] yes yes no yes no**

**Levels: no yes**

1. **The table function**

Now try the table() function in R. This function takes a factor variable and provides the number of occurrences of each category/level in that variable.

**x <- factor( c("yes", "yes", "no", "yes", "no") )**

**table(x)**

The print-out should be:

**no yes**

**2 3**

1. **The tapply() function**

The tapply() function is a mixture of the table and apply functions. The tapply function runs a function across two variables, for example you can run the mean function to calculate the average mark of people in each group. Try the following code:

**marks <- c(57,85,91,69,72,52)**

**group <- factor(c("A", "B", "B", "C", "A", "C"))**

**tapply(marks, group, mean)**

**tapply(marks, group, range)**

1. **Handling missing values**

Try the following code:

**x <- c(25,26,NA,56,42,25)**

**is.na(x)**

**is.nan(x)**

The function is.na(x) is used to test for NA and returns a logical vector.

The function is.nan(x) is used to test for NaN and returns a logical vector.

**x <- c(1, 2, NaN, NA, 4)**

**is.na(x)**

Print-out should be: [1] FALSE FALSE TRUE TRUE FALSE

**is.nan(x)**

Print-out should be: [1] FALSE FALSE TRUE FALSE FALSE

1. **Data cleansing of missing values**

Try the following two approaches for cleansing missing values. The first approach sets all missing values to 0 and the second just inputs the mean average.

**x <- c(1, 2, NA, 10, 3)**

**x[is.na(x) | is.nan(x) ] <- 0**

**x <- c(1, 2, NA, 10, 3)**

**x[is.na(x) | is.nan(x)] <- mean(x, na.rm=TRUE)**

1. **The mean function with missing values:**

If you try the following code, you will notice that it does not work as expected:

**x <- c(1, 2, NA, 10, 3)**

**mean(x)**

To work as expected you need to use the na.rm argument to remove NA values:

**mean(x, na.rm=TRUE)**

1. **Data Frames**

Data frames are like spreadsheets and can be created or imported using any of the following functions:

* + data.frame(), read.table() or read.csv()

Create your own data frame:

**x <- data.frame(names= c("ben", "joe", "steve"), ages= c(31, 24, 45))**

Now use the following functions to determine the number of columns and rows:

**nrow(x)**

**ncol(x)**

Now use the structure function – to learn more type **?str**

**str(x)**

Now you can access any column/variable in a data frame using the dollar sign. Try the following:

**x$names**

Note ‘x’ is the **name** of the data frame and ‘names’ is the name of the column.

Type in the following to see a list of free data frames in R:

**data()**

Type in the following:

**x <- mtcars**

A new dataframe **x** should appear in your environment. You can read about it by typing in the following:

**?mtcars**

Now have a go at analyzing this data.

Boxplot miles per gallon and create a histogram:

**boxplot(mtcars$mpg)**

**hist(mtcars$mpg)**

Boxplot miles per gallon for manual and automatic cars (variable **am** in dataset).

First method – sub-setting approach:

**par(mfrow=c(2,1))**

**boxplot(mtcars$mpg[mtcars$am==1], main="auto cars")**

**boxplot(mtcars$mpg[mtcars$am==0], main="manual cars")**

Reset opened par()

**dev.off()**

Boxplot miles per gallon for manual and automatic cars

Second method:

**boxplot(mtcars$mpg~mtcars$am)**