# 605_HW7.Rmd

*Kumudini Bhave*

*March 21, 2017*

---

## Expectation, Conditional Probability

---

**Problem Set 2 :**

**This week, you'll have only one programming assignment. Please write a function to compute the expected value and standard deviation of an array of values. Compare your results with that of R's mean and std functions. Please document your work in an R-Markdown file and ensure that you have good comments to help the reader follow your work. Now, consider that instead of being able to neatly fit the values in memory in an array, you have an infinite stream of numbers coming by. How would you estimate the mean and standard deviation of such a stream? Your function should be able to return the current estimate of the mean and standard deviation at any time it is asked. Your program should maintain these current estimates and return them back at any invocation of these functions. (Hint: You can maintain a rolling estimate of the mean and standard deviation and allow these to slowly change over time as you see more and more new values).**

**Solution :**

```
knitr::opts_chunk$set(message = FALSE, echo = TRUE)


# Library for data display in tabular format
library(DT)
```

```
#  Function to find the probabilities based on frequency of elements in the given array
# We remove the NA values to calculate the expected values /std programmatically as well as comparing w

calcProb <- function(dataArr){

     # dataArray <- c(1, 2, 3, 4, 5, 6, 3, 4, 2, 2, 1, 5)
     #dataArr <- c(1, 2, 3, 4, 5, 6, 3, 4, 2, 2, 1, NA)

     sumdata <- 0
     count <- 0
     dataArr <- na.omit(dataArr)

     # To find frequency of the elements in array
     dataTab <- table(dataArr)

     dataDF <- as.data.frame(dataTab)
```

```
     # Assign probability for each element of array

     Tot<- sum(dataDF$Freq)
     Tot
     dataDF$Prob <- dataDF$Freq / Tot

     return (dataDF)

}


# Function calculate the expected value
calcExpVal <- function(dMat){

     # Rows in total ie no of distinct elements
     numrow <- nrow(dMat)

     eval <- 0
     for(i in 1:numrow)
     {
          #paste(dMat[i,1] ," " ,   dMat[i,3])
          eval <- eval + (as.numeric(dMat[i,1]) * as.numeric(dMat[i, 3]))
     }
     return (round(eval, 2))
}

# Function calculate the  standard deviation

calcSTD <- function(dMat, expval){

     # Var(x) = E(x^2) - (E(x) )^2

     # Rows in total ie no of distinct elements
     numrow <- nrow(dMat)

     dvar <- 0

     # Calculating sum of  E(x^2) in this loop
     for(i in 1:numrow)
     {
          # paste(dMat[i,1] ," " ,   dMat[i,3])
          # E(x^2)
          dvar <- dvar + (as.numeric(dMat[i,1]) * as.numeric(dMat[i,1]) * as.numeric(dMat[i, 3]))

     }

     #E(x^2) - (E(x) )^2
     dvar <- dvar - (expval^2)
     stdval <- round(sqrt(dvar),2)

     return (stdval)
```

```
}


# Test Data 1
dataArray <- c(1, 2, 3, 4, 5, 6, 3, 4, 2, 2, 1, NA)
dframe <- calcProb(dataArray)
dMat <- as.matrix((dframe))
dMat
```

```
##      dataArr Freq Prob
## [1,] "1"     "2"  "0.18181818"
## [2,] "2"     "3"  "0.27272727"
## [3,] "3"     "2"  "0.18181818"
## [4,] "4"     "2"  "0.18181818"
## [5,] "5"     "1"  "0.09090909"
## [6,] "6"     "1"  "0.09090909"
```

```
dexp <- calcExpVal(dMat)
dexp
```

```
## [1] 3
```

```
dstd <- calcSTD(dMat, dexp)
dstd
```

```
## [1] 1.54
```

```
# Call r inbuilt function to calulate Mean of the data array

meanR <- mean(dataArray, na.rm=TRUE)
meanR
```

```
## [1] 3
```

```
stdR <- sd(dataArray, na.rm=TRUE)
stdR
```

```
## [1] 1.612452
```

```
wmeanR <- weighted.mean(dataArray, na.rm=TRUE)
wmeanR
```

```
## [1] 3
```

```
# Test Data 2

dataArray <- c(1, 2, 3, 4, 5, 6, 3, 4, 2, 2, 1, 5)
dframe <- calcProb(dataArray)
dMat <- as.matrix((dframe))
dMat
```

```
##      dataArr Freq Prob
## [1,] "1"     "2"  "0.16666667"
## [2,] "2"     "3"  "0.25000000"
## [3,] "3"     "2"  "0.16666667"
## [4,] "4"     "2"  "0.16666667"
## [5,] "5"     "2"  "0.16666667"
## [6,] "6"     "1"  "0.08333333"
```

```
dexp <- calcExpVal(dMat)
dexp
```

```
## [1] 3.17
```

```
dstd <- calcSTD(dMat, dexp)
dstd
```

```
## [1] 1.57
```

```
# Call r inbuilt function to calulate Mean of the data array

meanR <- mean(dataArray, na.rm=TRUE)
meanR
```

```
## [1] 3.166667
```

```
stdR <- sd(dataArray, na.rm=TRUE)
stdR
```

```
## [1] 1.642245
```

```
wmeanR <- weighted.mean(dataArray, na.rm=TRUE)
wmeanR
```

```
## [1] 3.166667
```

```
# Test Data 3
dataArray <- c(-1,-1,-1,0,0,0,0,3,3,5)
dframe <- calcProb(dataArray)
dMat <- as.matrix((dframe))
dMat
```

```
##      dataArr Freq Prob
## [1,] "-1"    "3"  "0.3"
## [2,] "0"     "4"  "0.4"
## [3,] "3"     "2"  "0.2"
## [4,] "5"     "1"  "0.1"
```

```
dexp <- calcExpVal(dMat)
dexp
```

```
## [1] 0.8
```

```
dstd <- calcSTD(dMat, dexp)
dstd
```

```
## [1] 1.99
```

```
# Call r inbuilt function to calulate Mean of the data array

meanR <- mean(dataArray, na.rm=TRUE)
meanR
```

```
## [1] 0.8
```

```
stdR <- sd(dataArray, na.rm=TRUE)
stdR
```

```
## [1] 2.097618
```

```
wmeanR <- weighted.mean(dataArray, na.rm=TRUE)
wmeanR
```

```
## [1] 0.8
```

We find that the mean and the expected value quite match

---

**find the mean and the std deviations for a stream of numbers, i.e. continuous variables**

```
roll_stat<-function(rollnum){

     roll_n <- 0
     roll_tot <- 0
     roll_mean <- 0
     roll_var <- 0
     roll_sd <- 0
     roll_sum_sq <- 0


     for (i in 1:length(rollnum))
     {
          # add each new element count
          roll_n <- roll_n + 1

          # add each element to element value sum
          roll_tot <- roll_tot + rollnum[i]

          # Add the sum of squares
          roll_sum_sq <- roll_sum_sq + rollnum[i]^2

          # Cal mean
          roll_mean <- roll_tot / roll_n

          # Cal std deviation
          roll_sd <- sqrt((roll_n)* roll_sum_sq - roll_tot^2)/(roll_n)
     }

     rollingstats <- c(roll_n, roll_mean, roll_sd)
     return (as.list(rollingstats))


}


# Test the infinites stream of numbers

infnum <- rnorm(n=1000, mean = 88, sd = 5)
roll_stat(infnum)
```

```
## [[1]]
## [1] 1000
##
```

```
## [[2]]
## [1] 87.77171
##
## [[3]]
## [1] 4.946463
```

```r
mean(infnum)
```

```
## [1] 87.77171
```

```r
sd(infnum)
```

```
## [1] 4.948938
```

```r
infnum1 <- sample(25, 1000,  replace = TRUE)
roll_stat(infnum1)
```

```
## [[1]]
## [1] 1000
##
## [[2]]
## [1] 12.882
##
## [[3]]
## [1] 7.262649
```

```r
mean(infnum1)
```

```
## [1] 12.882
```

```r
sd(infnum1)
```

```
## [1] 7.266283
```

```r
infnum2 <- sample(500, 1000,  replace = TRUE)
roll_stat(infnum2)
```

```
## [[1]]
## [1] 1000
##
## [[2]]
## [1] 240.95
##
## [[3]]
## [1] 143.9214
```

```r
mean(infnum2)
```

```
## [1] 240.95
```

```r
sd(infnum2)
```

```
## [1] 143.9934
```

We find that the calcualted rolling statistics for the infinite stream of numbers yields the mean and std deviation that matches that of built in R functions