BLin_Assign7

Bin Lin

2017-3-19

1. Problem Set 1 Please write a function to compute the expected value and standard deviation of an array of values.

$$E(x) = \sum_i P(x == i) \times i$$

```
expected_value <- function(v)
{
  n <- length(v)
  E <- 0

  #Based on the formula, it is undefined if vector is empty.
  if (n==0)
  {
    print ("Vector is empty")
  }
  else
  {
      for (i in v)
        {
        E <- E + (i / n)
        }
  }
  return (E)
}

v <- 1:10
expected_value(v)
```

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

Population Standard Deviation formula: $\sigma = \sqrt{(\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$

Sample Standard Deviation formula: $s = \sqrt{(\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$

```
standard_deviation <- function(v)
{
  n <- length(v)
  #Based on the sample standard deviation formula, it is undefined if vector has only one value.
  if (n == 1)
  {
    print ("Please enter vectors with more than 1 value")
  }
  else
  {
    variance <- (sum((v - expected_value(v)) ^ 2) / (n - 1))
    s <- sqrt(variance)
  }
  return (s)
}
v <- 1:10
standard_deviation(v)
```

```
## [1] 3.02765
```

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.

```
mean(v) == expected_value(v)
```

```
## [1] TRUE
```

```
sd(v) == standard_deviation(v)
```

```
## [1] TRUE
```

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).

There is another formula can be used to calculate population standard deviation. The formula is shown in the following. Proof can be found on the wikipedia website: https://en.wikipedia.org/wiki/Standard_deviation (https://en.wikipedia.org/wiki/Standard_deviation)

$$\sigma = \sqrt{E(x^2) - (E(x))^2}$$

```
#First of all, I am setting up all the global variables and set them equals to 0.
e <- 0
sum_xsquare <- 0
sigma <- 0
n <- 0

result <- data.frame(e, sigma)
names(result) <- c("Current Mean", "Current Standard Deviation")


rolling_estimate <- function(v)
{

  #Secondly, I am updating those global variables according to the formula above.
  e <- (n * e + sum(v)) / (n + length(v))
  n <- n + length(v)
  sum_xsquare <- sum_xsquare + sum(v ^ 2)
  sigma <- sqrt((sum_xsquare / n)  - (e ^ 2))
  result <- rbind(result, c(e, sigma))


  #Thirdly, use assign function to keep using these global variables at next invocation of this
 function.
  assign("e", e, envir = .GlobalEnv)
  assign("n", n, envir = .GlobalEnv)
  assign("sum_xsquare", sum_xsquare, envir = .GlobalEnv)
  assign("sigma", sigma, envir = .GlobalEnv)
  assign("result", result, envir = .GlobalEnv)
  return (result)
}

rolling_estimate(v)
```

```
##   Current Mean Current Standard Deviation
## 1          0.0                   0.000000
## 2          5.5                   2.872281
```

```
w <- 11:20

rolling_estimate(w)
```

```
##   Current Mean Current Standard Deviation
## 1          0.0                   0.000000
## 2          5.5                   2.872281
## 3         10.5                   5.766281
```

```
x <- rnorm(1000, mean = 10, sd = 5)
rolling_estimate(x)
```

```
##   Current Mean Current Standard Deviation
## 1      0.00000                   0.000000
## 2      5.50000                   2.872281
## 3     10.50000                   5.766281
## 4     10.15167                   5.309286
```

```
y <- rnorm(10000, mean = 10, sd = 5)
rolling_estimate(y)
```

```
##   Current Mean Current Standard Deviation
## 1     0.000000                   0.000000
## 2     5.500000                   2.872281
## 3    10.500000                   5.766281
## 4    10.151675                   5.309286
## 5     9.974198                   5.071386
```