ES207 HW1

Erik Bolch

3. What happens if your try to print out the 4th element of x? Print your result and provide the answer in your lab write up.

```
x<-c(1,2,4)
x
```

```
## [1] 1 2 4
```

```
print(x)
```

```
## [1] 1 2 4
```

```
x[3]
```

```
## [1] 4
```

```
x[2:3]
```

```
## [1] 2 4
```

```
x[4]
```

```
## [1] NA
```

```
print(x[4])
```

```
## [1] NA
```

There is nothing indexed at position 4, so NA is returned.

4. Try creating a variable "s" that is the standard deviation of q. Make sure you print it out to confirm it worked.

```
q <- c(x,x,8)
# Mean of the x vector
mean(x)
```

```
## [1] 2.333333
```

```
# Standard deviation of the x vector
sd(x)
```

```
## [1] 1.527525
```

```r
y <- mean(x)
# And then print it by:
y
```

```
## [1] 2.333333
```

```r
stdev<-sd(q)
stdev
```

```
## [1] 2.478479
```

5. Can you make R write your name? Print the Results to confirm that it worked.

```r
paste("Erik","Bolch")
```

```
## [1] "Erik Bolch"
```

```r
#Assign R objects
a <- 1+1
b <- 24/12
c <- 100^2
#Perform some math on the objects
d=(a+c)/b
#Print the results
d
```

```
## [1] 5001
```

```r
# print out y
print(y)
```

```
## [1] 2.333333
```

```r
y
```

```
## [1] 2.333333
```

```r
# assign value 100 to object "m"
m <- 100
# list objects in my current environment
ls()
```

```
## [1] "a"     "b"     "c"     "d"     "m"     "q"     "stdev" "x"     "y"
```

```r
rm(m)
```

6. What objects are left in your R session after removing m?

```
ls()
```

```
## [1] "a"     "b"     "c"     "d"     "q"     "stdev" "x"     "y"
```

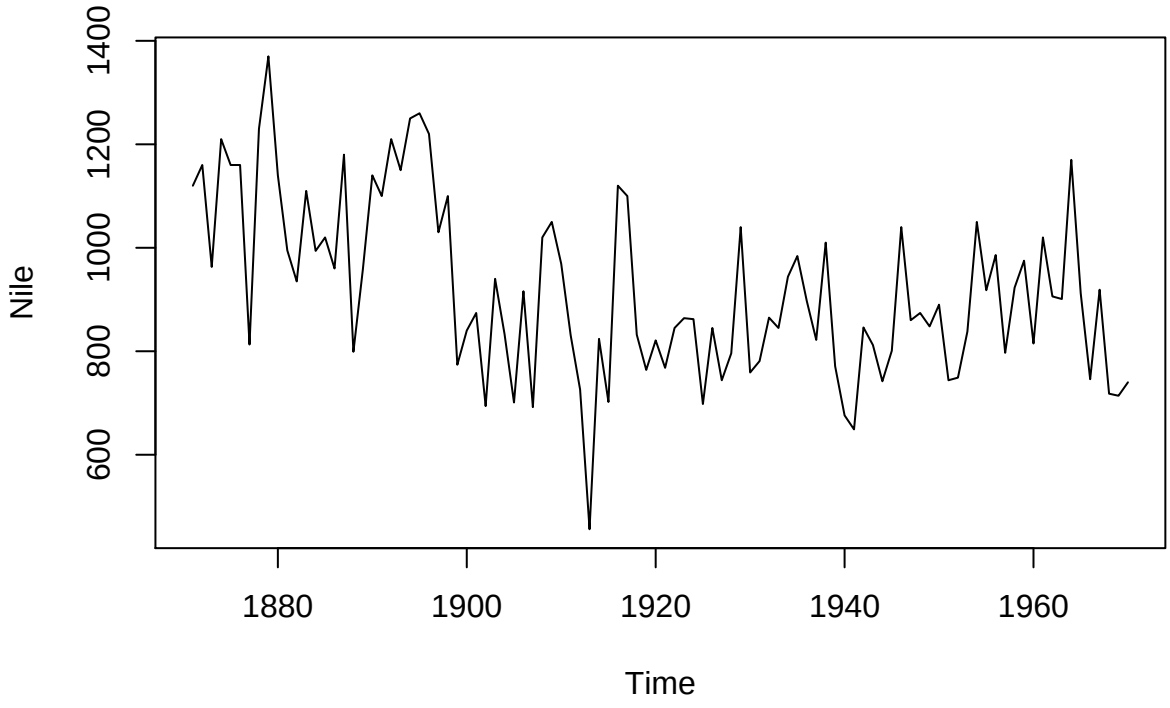
```
rm(list=ls())
ls()
```

```
## character(0)
```

```
Nile
```

```
## Time Series:
## Start = 1871
## End = 1970
## Frequency = 1
##   [1] 1120 1160  963 1210 1160 1160  813 1230 1370 1140  995  935 1110  994
##  [15] 1020  960 1180  799  958 1140 1100 1210 1150 1250 1260 1220 1030 1100
##  [29]  774  840  874  694  940  833  701  916  692 1020 1050  969  831  726
##  [43]  456  824  702 1120 1100  832  764  821  768  845  864  862  698  845
##  [57]  744  796 1040  759  781  865  845  944  984  897  822 1010  771  676
##  [71]  649  846  812  742  801 1040  860  874  848  890  744  749  838 1050
##  [85]  918  986  797  923  975  815 1020  906  901 1170  912  746  919  718
##  [99]  714  740
```

```
plot(Nile)
```



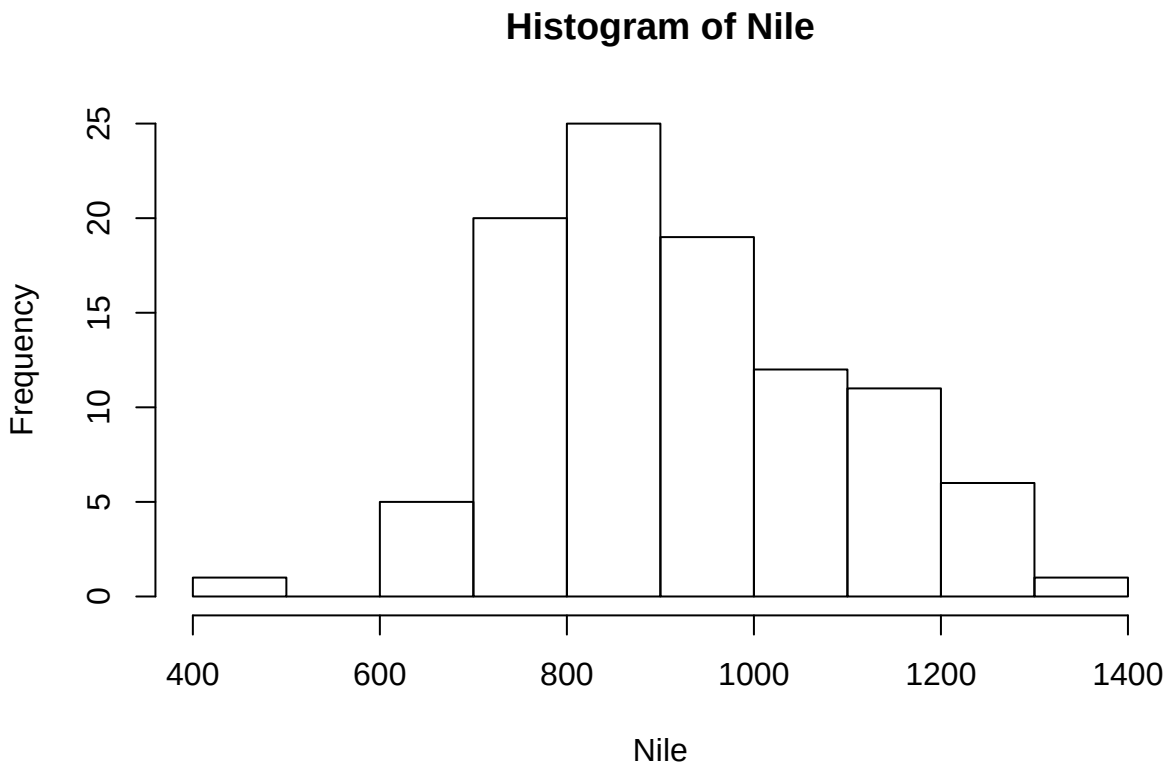
```
meanNile <- mean(Nile)
meanNile
```

```
## [1] 919.35
```

```
sdNile <- sd(Nile)
sdNile
```

```
## [1] 169.2275
```

```
hist(Nile)
```




```
oddcount <- function(x)
{
    k <- 0 # assign 0 to k
    for (n in x) {
        if(n %% 2 == 1)
        {
            k <- k+1 # %% is the modulo operator
        }
    }
  return(k)
}
oddcount
```

```
## function(x)
## {
##     k <- 0 # assign 0 to k
##     for (n in x) {
##         if(n %% 2 == 1)
##         {
##             k <- k+1 # %% is the modulo operator
##         }
##     }
##   return(k)
## }
```

```
oddcount(x <- c(1,3,5))
```

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

```
oddcount(x <- c(1,2,3,7,9))
```

```
## [1] 4
```

7. How many odd numbers were in this vector?. 4

```
38 %% 7
```

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

```
38 %% 2
```

```
## [1] 0
```

```
39 %% 2
```

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

```
y <- c(3,0,7)
for(n in y) { print(n) } # Print simply prints the value of the variable
```

```
## [1] 3
## [1] 0
## [1] 7
```

```
n <- y[1]
print(n)
```

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

```r
n <- y[2]
print(n)
```

```
## [1] 0
```

```r
n <- y[3]
print(n)
```

```
## [1] 7
```

```r
37 %% 2
```

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

```r
37 %% 2 == 1
```

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

```r
38 %% 2
```

```
## [1] 0
```

```r
38 %% 2 == 1
```

```
## [1] FALSE
```

```r
oddcount <- function(x) {
#   print("x is:")
    print(x)
    k <- 0 # assign 0 to k
    print(paste("k is initialized as",k))
    for (n in x) {
        print(paste("current x value being tested is",n))
        if(n %% 2 == 1)
        {
            k <- k+1 # %% is the modulo operator
            print(paste(n,"is an odd number!"))
        } else
        {
            print(paste(n,"is an even number!"))
        }
        print(paste("k is currently",k))
    }
    print(paste("The final k is",k))
    return(k)
}
# And trying running our more verbose function:
oddcount(x <- c(1,2,3,7,9))
```

```
## [1] 1 2 3 7 9
## [1] "k is initialized as 0"
## [1] "current x value being tested is 1"
## [1] "1 is an odd number!"
## [1] "k is currently 1"
## [1] "current x value being tested is 2"
## [1] "2 is an even number!"
## [1] "k is currently 1"
## [1] "current x value being tested is 3"
## [1] "3 is an odd number!"
## [1] "k is currently 2"
## [1] "current x value being tested is 7"
## [1] "7 is an odd number!"
## [1] "k is currently 3"
## [1] "current x value being tested is 9"
## [1] "9 is an odd number!"
## [1] "k is currently 4"
## [1] "The final k is 4"

## [1] 4
```

8. Try creating a new function "evencount" that counts the even numbers in a vector. Turn in your script as a .R script. Make sure you add appropriate comments - you will be graded on this.

```
evencount <- function(x)
{
    k <- 0 # assign 0 to k
    for (n in x) {
        if(n %% 2 == 0)
        {
            k <- k+1 # %% is the modulo operator
        }
    }
  return(k)
}
evencount(c(2,4,6,7,8,9,24))
```

```
## [1] 5
```

9. What are the three main reasons you want to have a good project layout? Can you think of any others?

The three main reasons to keep a good project layout are: able maintain data integrity, project portability, and it makes it easy to pick up after a break. Good project layout also allows people to collaborate more easily.

10. What are the three primary principles to follow in a good project layout?

Three primary principles to follow in good project layout are: -Treat data as read only -Treat generated output as disposible -Separate function definition and application

11. Write out the full path for your R installation. Use the format of the operating system you are currently using.

"C:\Program Files\R\R-3.6.2\"

12. Write the path above using a different operating system.

Linux file structure: "/Program Files/R/R-3.6.2/"

13. Write out the full path for the directory structure you have set up for this class all the way to where you have saved this .html tutorial.

Z:/Erik/ES207/HW1/

14. Write out the relative path for this .html file assuming your working directory to be set to your equivalent of Users/CardiB/classes/.

/ES207/HW1/

15. Write out the paths in an operating system other than your own.

/beanstore/Erik/ES207/HW1/

16. Complete Steps 1-28 in the "Happy Git with R" tutorial.

17. In step 9 install SourceTree. Note: If you're using Google Chrome with a google account, and also use a google account for registering with Atlassian it makes life easy. You will not need to worry about setting up HTTPS or SSH authorization to make push and pull requests.

18. Make a repo on GitHub called "ES207_hw1. Write the render-read R script in 20.2 and commit it to your repo.

https://github.com/ebolch/es207_hw1/blob/master/render-ready.R

19. Following the instructions in 23, clone a GitHub repository that interests you. Find one simple script in the repo and run the code locally and understand what it is doing. Once you understand what it is doing, describe it in your Word doc and commit it to your hw1 repo. Make sure to cite the source of your script, and document any changes you made. DO NOT fork this code (we will do that next time).

https://github.com/ebolch/es207_hw1/blob/master/19_clone_a_repo.R

This script uses multiple datasets to calculate gun deaths per 100,000 people by cause and demographics.

20. Following the instructions in 25, fork the 'bingo' repo. Clone it to your local machine and create a new bingo card. Commit and push your changes back to your copy of the repo on GitHub. Make a pull request back to the main 'bingo' repo. Turn in your new bingo card.

https://github.com/ebolch/es207_hw1/blob/master/bingo_card.R

21. Provide your GitHub account name here.

https://github.com/ebolch