Data Science Practicals

Sunny Mishra | 2K17/CS/19

5/18/2020

Practical 1

Write a program that prints 'Hello World' to the screen.

```
print("hello, world!")
```

```
## [1] "hello, world!"
```

Practical 2

Write a program that asks the user for a number n and prints the sum of the numbers 1 to n

```
# n = as.integer(readline("Enter n: "))
n = 10
print(paste("Sum =",sum(1:n)))
```

```
## [1] "Sum = 55"
```

Practical 3

Write a program that prints a multiplication table for numbers up to 12.

```
for(i in 1:12) {
  print(paste("table of", i))
  for(j in 1:10) {
    print(paste(i,"x",j,"=",i*j))
  }
  print("")
}
```

```
## [1] "table of 1"
## [1] "1 x 1 = 1"
## [1] "1 x 2 = 2"
## [1] "1 x 3 = 3"
## [1] "1 x 4 = 4"
## [1] "1 x 5 = 5"
## [1] "1 x 6 = 6"
## [1] "1 x 7 = 7"
## [1] "1 x 8 = 8"
## [1] "1 x 9 = 9"
## [1] "1 x 10 = 10"
## [1] ""
## [1] "table of 2"
```

```
## [1] "2 x 1 = 2"
## [1] "2 x 2 = 4"
## [1] "2 x 3 = 6"
## [1] "2 x 4 = 8"
## [1] "2 x 5 = 10"
## [1] "2 x 6 = 12"
## [1] "2 x 7 = 14"
## [1] "2 x 8 = 16"
## [1] "2 x 9 = 18"
## [1] "2 x 10 = 20"
## [1] ""
## [1] "table of 3"
## [1] "3 x 1 = 3"
## [1] "3 x 2 = 6"
## [1] "3 x 3 = 9"
## [1] "3 x 4 = 12"
## [1] "3 x 5 = 15"
## [1] "3 x 6 = 18"
## [1] "3 x 7 = 21"
## [1] "3 x 8 = 24"
## [1] "3 x 9 = 27"
## [1] "3 x 10 = 30"
## [1] ""
## [1] "table of 4"
## [1] "4 x 1 = 4"
## [1] "4 x 2 = 8"
## [1] "4 x 3 = 12"
## [1] "4 x 4 = 16"
## [1] "4 x 5 = 20"
## [1] "4 x 6 = 24"
## [1] "4 x 7 = 28"
## [1] "4 x 8 = 32"
## [1] "4 x 9 = 36"
## [1] "4 x 10 = 40"
## [1] ""
## [1] "table of 5"
## [1] "5 x 1 = 5"
## [1] "5 x 2 = 10"
## [1] "5 x 3 = 15"
## [1] "5 x 4 = 20"
## [1] "5 x 5 = 25"
## [1] "5 x 6 = 30"
## [1] "5 x 7 = 35"
## [1] "5 x 8 = 40"
## [1] "5 x 9 = 45"
## [1] "5 x 10 = 50"
## [1] ""
## [1] "table of 6"
## [1] "6 x 1 = 6"
## [1] "6 x 2 = 12"
## [1] "6 x 3 = 18"
## [1] "6 x 4 = 24"
## [1] "6 x 5 = 30"
## [1] "6 x 6 = 36"
```

```
## [1] "6 x 7 = 42"
## [1] "6 x 8 = 48"
## [1] "6 x 9 = 54"
## [1] "6 x 10 = 60"
## [1] ""
## [1] "table of 7"
## [1] "7 x 1 = 7"
## [1] "7 x 2 = 14"
## [1] "7 x 3 = 21"
## [1] "7 x 4 = 28"
## [1] "7 x 5 = 35"
## [1] "7 x 6 = 42"
## [1] "7 x 7 = 49"
## [1] "7 x 8 = 56"
## [1] "7 x 9 = 63"
## [1] "7 x 10 = 70"
## [1] ""
## [1] "table of 8"
## [1] "8 x 1 = 8"
## [1] "8 x 2 = 16"
## [1] "8 x 3 = 24"
## [1] "8 x 4 = 32"
## [1] "8 x 5 = 40"
## [1] "8 x 6 = 48"
## [1] "8 x 7 = 56"
## [1] "8 x 8 = 64"
## [1] "8 x 9 = 72"
## [1] "8 x 10 = 80"
## [1] ""
## [1] "table of 9"
## [1] "9 x 1 = 9"
## [1] "9 x 2 = 18"
## [1] "9 x 3 = 27"
## [1] "9 x 4 = 36"
## [1] "9 x 5 = 45"
## [1] "9 x 6 = 54"
## [1] "9 x 7 = 63"
## [1] "9 x 8 = 72"
## [1] "9 x 9 = 81"
## [1] "9 x 10 = 90"
## [1] ""
## [1] "table of 10"
## [1] "10 x 1 = 10"
## [1] "10 x 2 = 20"
## [1] "10 x 3 = 30"
## [1] "10 x 4 = 40"
## [1] "10 x 5 = 50"
## [1] "10 x 6 = 60"
## [1] "10 x 7 = 70"
## [1] "10 x 8 = 80"
## [1] "10 x 9 = 90"
## [1] "10 x 10 = 100"
## [1] ""
## [1] "table of 11"
```

```
## [1] "11 x 1 = 11"
## [1] "11 x 2 = 22"
## [1] "11 x 3 = 33"
## [1] "11 x 4 = 44"
## [1] "11 x 5 = 55"
## [1] "11 x 6 = 66"
## [1] "11 x 7 = 77"
## [1] "11 x 8 = 88"
## [1] "11 x 9 = 99"
## [1] "11 x 10 = 110"
## [1] ""
## [1] "table of 12"
## [1] "12 x 1 = 12"
## [1] "12 x 2 = 24"
## [1] "12 x 3 = 36"
## [1] "12 x 4 = 48"
## [1] "12 x 5 = 60"
## [1] "12 x 6 = 72"
## [1] "12 x 7 = 84"
## [1] "12 x 8 = 96"
## [1] "12 x 9 = 108"
## [1] "12 x 10 = 120"
## [1] ""
```

Practical 4

Write a function that returns the largest element in a list.

```
getMaxFromList <- function(l) {
  return(max(unlist(l)))
}
l <- list(c(10,2,1,2,3,9,3,2,4,3,1))
getMaxFromList(l)
```

```
## [1] 10
```

Practical 5

Write a function that computes the running total of a list.

```
sumList <- function(l) {
  return(sum(unlist(l)))
}

sumList(list(1:20))
```

```
## [1] 210
```

Practical 6

Write a function that tests whether a string is a palindrome.

```
isPalindrome <- function(str) {
  library(stringr)
```

```r
  str <- unlist(str_split(str, ""))
  revStr <- rev(str)
  for(i in 1:length(str)) {
    if (revStr[i] != str[i]) {
      return(FALSE)
    }
  }
  return(TRUE)
}
isPalindrome("1211")
```

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

Practical 7

Implement linear search.

```r
linSearch <- function(arr, target) {
  for(el in arr) {
    if (el == target) {
      return(TRUE)
    }
  }
  return(FALSE)
}
linSearch(1:20, 5)
```

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

Practical 8

Implement binary search.

```r
binSearch <- function(arr, target) {
  start <- 1
  end <- length(arr)
  while(start <= end) {
    mid <- as.integer((start+end)/2)
    if(arr[mid] == target) {
      return(TRUE)
    }
    else if(target < arr[mid]) {
      end <- mid-1
    } else {
      start <- mid+1
    }
  }
}

binSearch(1:20, 5)
```

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

Practical 9

Implement matrices addition , subtraction and Multiplication

```
matrixAdd <- function(A, B) {
  return(A+B)
}

matrixSub <- function(A, B) {
  return(A-B)
}

matrixMul <- function(A, B) {
  return(A%*%B)
}

A <- matrix(1:9, 3, 3)
B <- matrix(-1:-9, 3, 3)

matrixAdd(A, B)
```

```
##      [,1] [,2] [,3]
## [1,]    0    0    0
## [2,]    0    0    0
## [3,]    0    0    0
```

```
matrixSub(A, B)
```

```
##      [,1] [,2] [,3]
## [1,]    2    8   14
## [2,]    4   10   16
## [3,]    6   12   18
```

```
matrixMul(A, B)
```

```
##      [,1] [,2] [,3]
## [1,]  -30  -66 -102
## [2,]  -36  -81 -126
## [3,]  -42  -96 -150
```

Practical 10

Fifteen students were enrolled in a course. There ages were: 20 20 20 20 20 21 21 21 22 22 22 22 23 23 23
i. Find the median age of all students under 22 years
ii. Find the median age of all students
iii. Find the mean age of all students
iv. Find the modal age for all students
v. Two more students enter the class. The age of both students is 23. What is now mean, mode and median ?

```
age <- c(rep(20,5),rep(21,3),rep(22,4),rep(23,3))

median(age[age < 22])
```

```
## [1] 20
```

```
median(age)
```

```
## [1] 21
```

```
mean(age)
```

```
## [1] 21.33333
```

```
names(table(age))[table(age)==max(table(age))]
```

```
## [1] "20"
```

```
age = c(age, rep(23, 2))

mean(age)
```

```
## [1] 21.52941
```

```
median(age)
```

```
## [1] 22
```

```
names(table(age))[table(age)==max(table(age))]
```

```
## [1] "20" "23"
```

Practical 11

Following table gives a frequency distribution of systolic blood pressure. Compute all the measures of dispersion.

Midpoint	Number
95.5	5
105.5	8
115.5	22
125.5	27
135.5	17
145.5	9
155.5	5
165.5	5
175.5	2

Measures of dispersion:
they represent that how much the data is dispersed
these are of 2 types: ABSOLUTE & RELATIVE measures of dispersion
these can be represented in terms of: 1. Range 2. Standered Deviation 3. Variance 4. Quartiles and Quartile Deviation 5. Mean and Mean Deviation

```
systolic.blood.pressure.Midpoint = c(rep(95.5,5),
                                     rep(105.5,8),
                                     rep(115.5,22),
                                     rep(125.5,27),
                                     rep(135.5,17),
                                     rep(145.5,9),
                                     rep(155.5, 5),
                                     rep(165.5,5),
                                     rep(175.5,2))
# finding range
```

```r
range <- c(min(systolic.blood.pressure.Midpoint), max(systolic.blood.pressure.Midpoint))
range
```

```
## [1]  95.5 175.5
```

```r
meanValue <- mean(systolic.blood.pressure.Midpoint)  # mean
meanValue
```

```
## [1] 128.2
```

```r
sd(systolic.blood.pressure.Midpoint) # Standard deviation
```

```
## [1] 17.97051
```

```r
quartile <- quantile(systolic.blood.pressure.Midpoint) # Quartile
quartile
```

```
##    0%   25%   50%   75%  100%
##  95.5 115.5 125.5 135.5 175.5
```

```r
ul <- unique(systolic.blood.pressure.Midpoint)

# mean derviation
print("Term : Mean Deviation")
```

```
## [1] "Term : Mean Deviation"
```

```r
for(i in 1:length(ul)) {
  cat(ul[i], " : ", ul[i]-meanValue)
}
```

```
## 95.5  :  -32.7105.5  :  -22.7115.5  :  -12.7125.5  :  -2.7135.5  :  7.3145.5  :  17.3155.5  :  27.31
```

```r
# Quartile Deviation
Q1 = quartile[[1]]
Q3 = quartile[[3]]
quartileDev = (Q3-Q1)/(Q3+Q1)
quartileDev
```

```
## [1] 0.1357466
```

Practical 12

Obtain probability distribution of , where X is number of spots showing when a six-sided symmetric die (i.e. all six faces of the die are equally likely) is rolled. Simulate random samples of sizes 40, 70 and 100 respectively and verify the frequency interpretation of probability.

```r
s1 = sample(1:6, 40, replace = TRUE)
s2 = sample(1:6, 70, replace = TRUE)
s3 = sample(1:6, 100, replace = TRUE)

probFreq = table(s1)
probFreq
```

```
## s1
## 1 2 3 4 5 6
## 6 7 4 9 5 9
```

```r
probFreq = table(s2)
probFreq
```

```
## s2
##  1  2  3  4  5  6
## 17  9 14  6  8 16
```

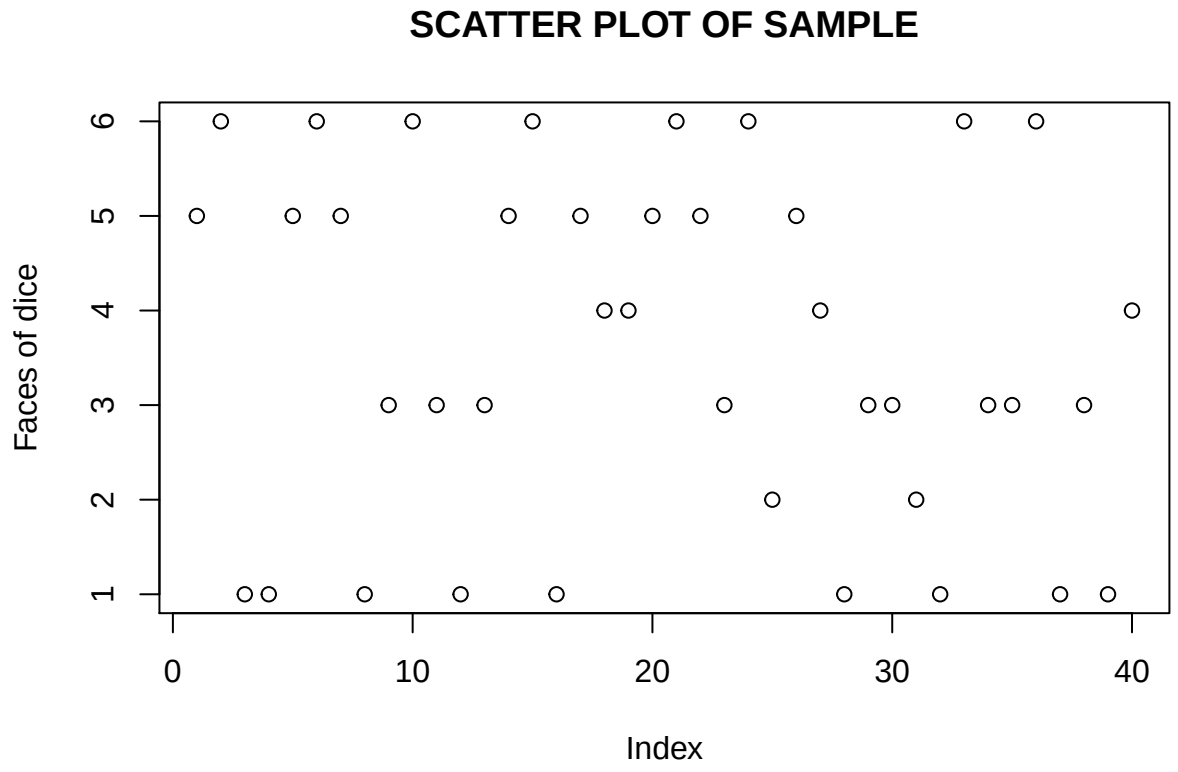
```r
probFreq = table(s3)
probFreq
```

```
## s3
##  1  2  3  4  5  6
## 23 17 16 14 17 13
```

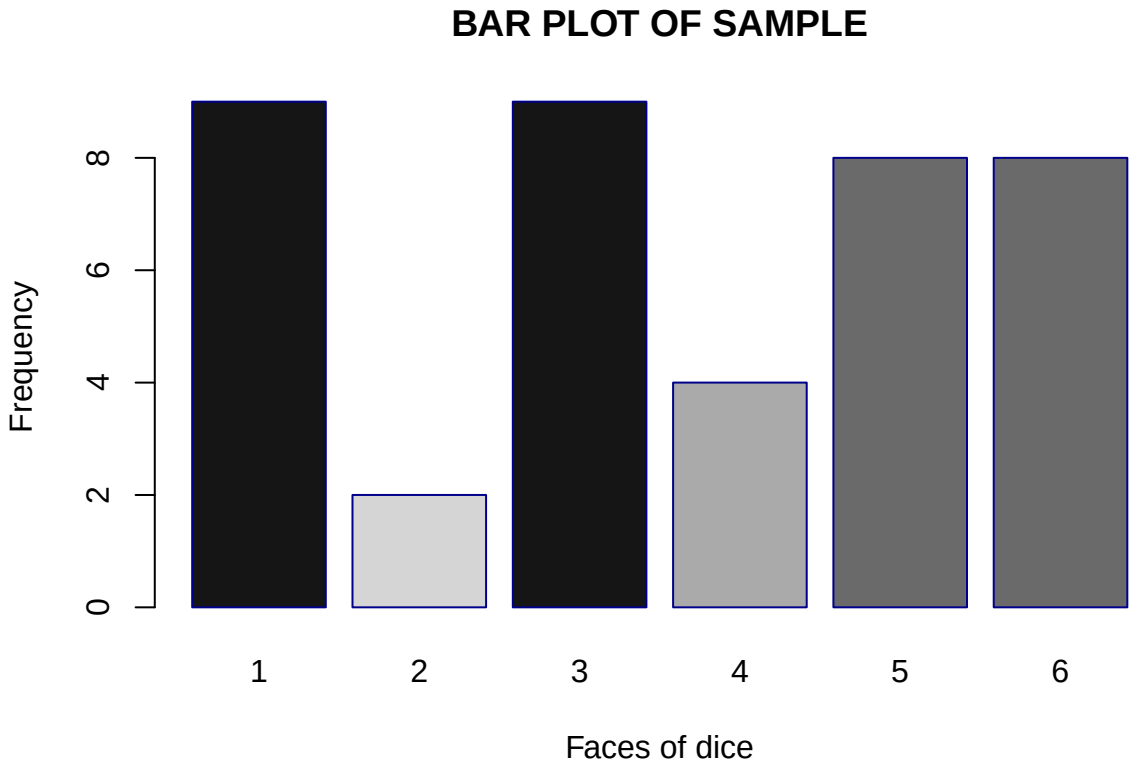
Practical 13

Make visual representations of data using the base, lattice, and ggplot2 plotting systems in R, apply basic principles of data graphics to create rich analytic graphics from available datasets.

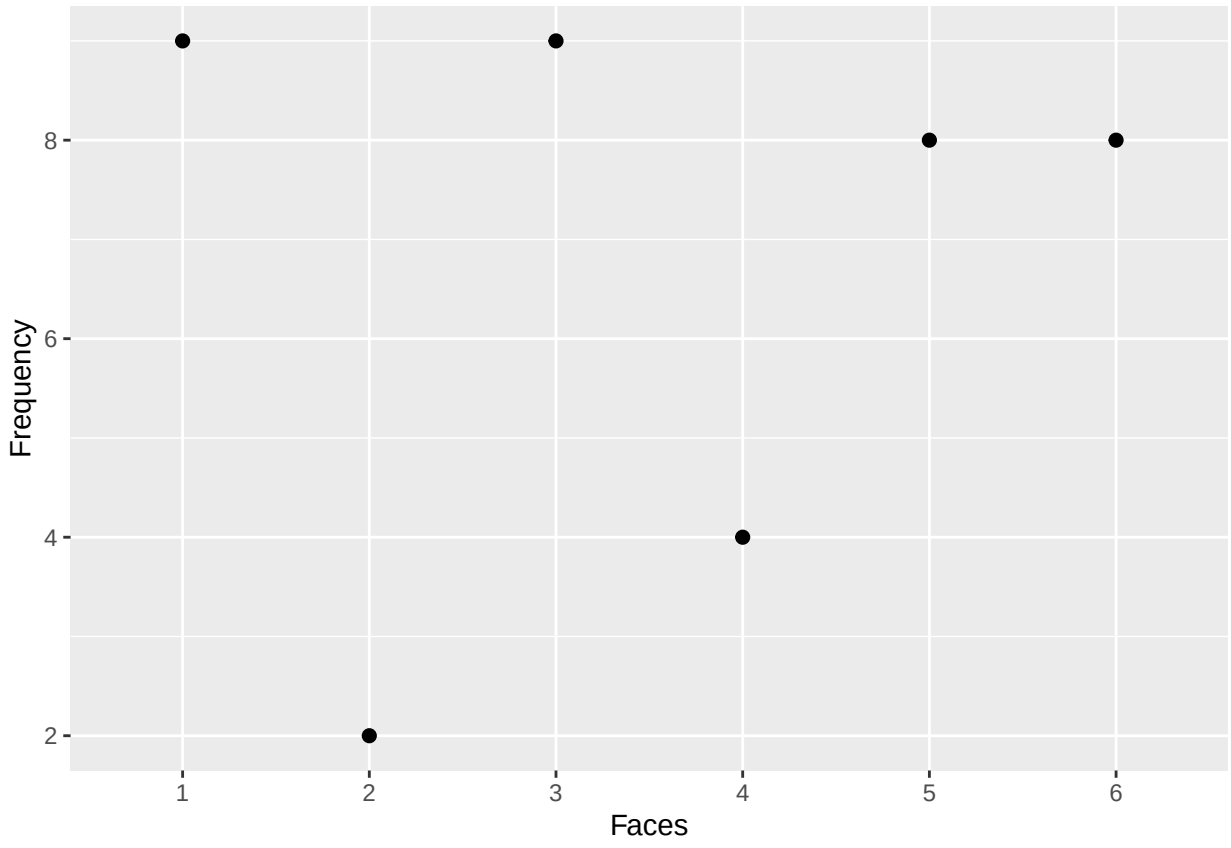
```r
library(ggplot2)

# random semple
s1 = sample(1:6, 40, replace = TRUE)
s2 = sample(1:6, 70, replace = TRUE)
s3 = sample(1:6, 100, replace = TRUE)

probFreq = table(s1)

probFreqDF  = as.data.frame(probFreq)
colnames(probFreqDF) = c("Faces", "Frequency")

rel = rank(table(s1))/length(table(s1))

### Plotting
plot(s1, main = "SCATTER PLOT OF SAMPLE",
     xlab = "Index",
     ylab = "Faces of dice")
```



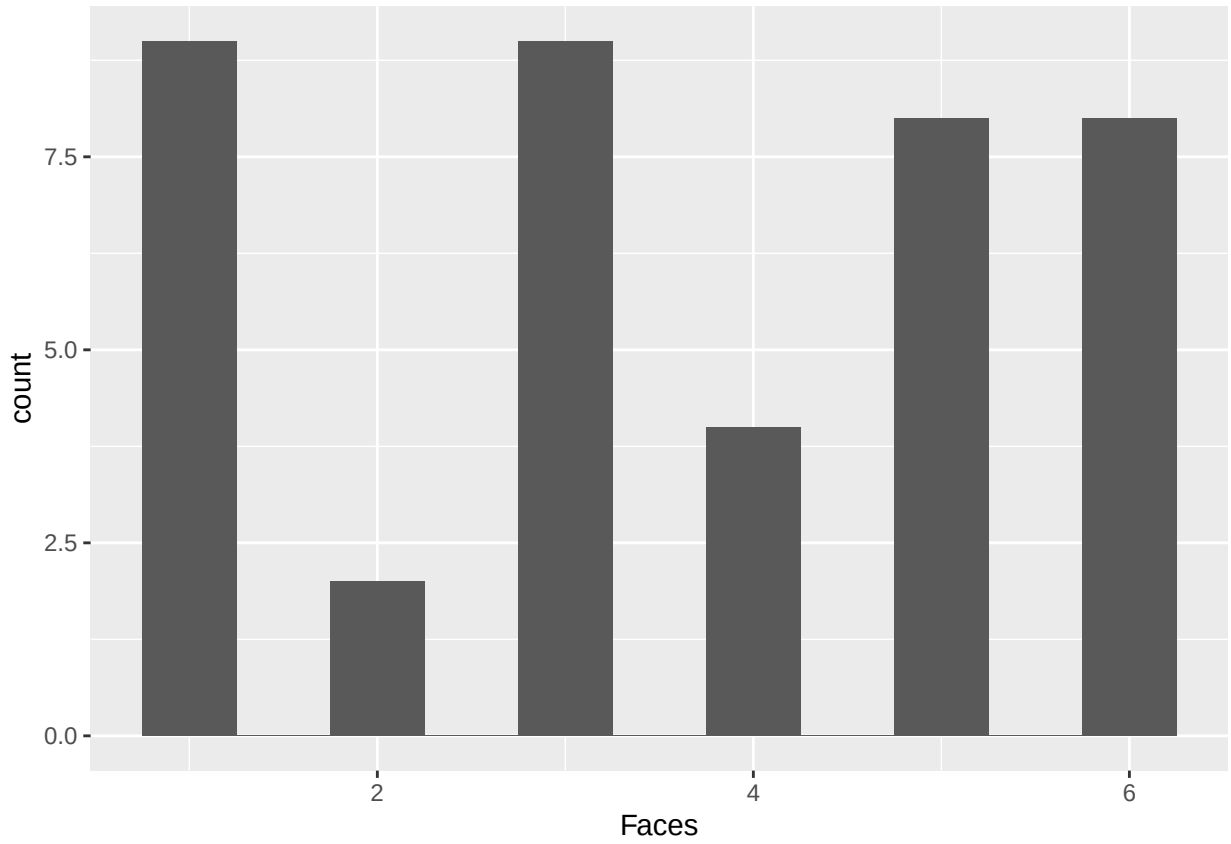

```
barplot(probFreq, main = "BAR PLOT OF SAMPLE",
        xlab = "Faces of dice",
        ylab = "Frequency",
        border = "dark blue",
        col = gray(1-rel))
```



```r
points = ggplot(probFreqDF,
                aes(x=Faces, y = Frequency)) + geom_point(size = 2)

##########################################
Faces = factor(NULL, levels = probFreqDF$Faces)
Freq = as.numeric(NULL)

for(i in 1:nrow(probFreqDF)) {
  faces = as.numeric(probFreqDF$Faces[i])

  val = rep(faces, as.numeric(probFreqDF$Frequency[i]))
  Faces = c(Faces, val)
  Freq = c(Freq, rep(1, as.numeric(probFreqDF$Frequency[i])))
}
temp = data.frame(Faces = Faces, Freq = Freq)

#  histogram
hist = ggplot(temp, aes(x=Faces)) +
          geom_histogram(binwidth = 0.5)

#################

points
```




```r
hist
```


count
7.5
5.0
2.5
0.0
2
4
6
Faces