# MA213 Basic Statistics and Probability - Lab4 guide

## Lab 4: Simulation and Probability R guide

---

## Learning Objectives

- Validate and Explain Probability Distributions: Assess the validity of a probability distribution using the concepts of outcome, sample space, and probability properties (e.g., disjoint outcomes, probabilities between 0 and 1, and total probabilities summing to 1).
- Compute Probabilities Using Various Tools: Use logic, Venn diagrams, and probability rules to compute probabilities for events.
- Understand and Compute Expectations and Variances: Explain the concepts of expectations and variances of random variables, and compute the expectation and variance of a linear combination of random variables.
- Conduct Hypothesis Testing Using Simulation: Set up null and alternative hypotheses to test for independence between variables, and use simulation techniques to evaluate data support for these hypotheses.

## How do you roll a die using R?

`sample()` will randomly sample `size` many sample from the vector `x`

`sample(x, size, replace = FALSE, prob = NULL)`

```
# example
x <- c("apple", "pear", "strawberry", "orange", "lemon")

sample(x, 3) # picks 3 unique fruits from the vector (without replacement)
```

```
## [1] "orange" "pear"   "apple"
```

```
sample(x, 10, replace=TRUE) # picks 10 fruits, allowing repeats (sampling with replacement)
```

```
##  [1] "lemon"      "strawberry" "orange"     "pear"       "lemon"
##  [6] "pear"       "lemon"      "lemon"      "apple"      "pear"
```

**Exercise : Let's make a die (a vector of size 6) and roll once.**

```
#
#
#
```

## Loops

Loops repeatedly execute a block of code for various elements.

```
# Example: Print numbers from 1 to 5
print("Example1 ")
```

```
## [1] "Example1 "
```

```
for (i in 1:5) {
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

```
# Example 2: Print numbers from 1 to 5
print("Example2 ")
```

```
## [1] "Example2 "
```

```
for (i in c(1,2,3,4,5)) {
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

```
# Example3: Simulate rolling a die 5 times and record it
print("Example3 ")
```

```
## [1] "Example3 "
```

```
rolls <- rep(0, 5) # Initialize a vector of length 5 with zeros to store roll results
for (i in 1:5) {
  rolls[i] <- sample(1:6, 1)    # Sample one number between 1 and 6 and assign it to the i-th position
}
rolls
```

```
## [1] 1 1 5 4 3
```

## Formal explanation

```
# For loop syntax
for (variable in sequence) {
    # code you want to execute repeatedly until the end of the sequence
}
```

**Exercise : Let's make a coin (a vector of size 2) and flip it 10 times and record them.**

```
#
#
#
x <- c(0,1)
sample(x,1)
```

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

```r
rolls <- rep(0,10)
for (i in 1:10){
  rolls[i] <- sample(x,1)
}

print(rolls)
```

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

### Function

Functions allow you to encapsulate reusable blocks of code and parameters.

```r
# Example: Function to calculate (a+b)^2
calc <- function(a,b){
  output <- a+b
  return(output)
}
```

```r
calc <- function(#argument for input){

  return(output) #output can be a value or a parameter
                  # or you don't need an output sometimes
}
```

```r
# input or output not necessary
greet <- function(){
  print("Hello there")
}

greet()
```

```
## [1] "Hello there"
```

```r
greet()
```

```
## [1] "Hello there"
```

```r
greet()
```

```
## [1] "Hello there"
```

#### Exercise : Make a function to calculate the multiplication of the three numbers

```r
# what is your function name?
# what should be inputs?
# what is the output?
```

### Function + Loop

Combine loops and functions for repeated tasks.

```r
# Example: Roll a die 5 times using a loop within a function
roll_multiple <- function(runs) {
  rolls <- rep(0, runs) # create rolls vector of 0

   for (i in 1:runs) {
    rolls[i] <- sample(1:6, 1)
  }

  return(rolls)
}
roll_multiple(5)
```

```
## [1] 1 3 6 3 1
```

### Let's roll one die and show the output.

```r
#Outcome from those two dice

die1 <- sample(1:6, 1)
die2 <- sample(1:6, 1)

sum_of_two <- die1 + die2
sum_of_two
```

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

### Using `for loop`, assign outcome of two dice rolled into `myoutcome` from 100 iterations.

```r
myoutcome = rep(0, 100) # make a vector of size 100 that has all zero's

for (i in 1:100){
  dice1 <- sample(1:6, 1)
  dice2 <- sample(1:6, 1)
  sum_of_two <- dice1 + dice2
  myoutcome[i] <- sum_of_two
}

head(myoutcome) # print first 6 entries
```

```
## [1] 4 8 7 2 9 5
```

### Use `replicate()`, it carries out repeated tasks in a more computationally efficient way.

```r
twodice_outcome <- function(){
  dice1 <- sample(1:6, 1)
  dice2 <- sample(1:6, 1)

  sum_of_two <- dice1 + dice2
  myoutcome <- sum_of_two

  return(myoutcome)
}
twodice_outcome() # this function works as one simulation run
```

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

```r
myoutcome <- replicate(n=30, twodice_outcome()) # 10 runs
myoutcome
```

```
##  [1]  7  2 11  6  7  7  6  4  3  9  6  8  5  6  4  5  7  4  9  7  6  5  5  2  9
## [26]  7  7 12 11  9
```

### Calculate probability of each outcome.

```r
# for loop version (clasic)
n = 30 # assign 30 for rolling times
myoutcome <- rep(0, n) #initialize empty vector or size n
for (i in 1:n){
  myoutcome[i] <- twodice_outcome() # store each outcome to ith entry in myoutcome vector
}
myoutcome # print out myoutcome
```

```
##  [1] 12  4  5 12  8  7  7  5  9  4  6 10  9  8  4 11  4  7 10  2 10  2  7  7  8
## [26]  7  9  7 11  8
```

```r
outcome_table <- table(myoutcome)
n <- length(myoutcome) # this length function will count how many unique output you have.
prob_outcome <- outcome_table/n # to calculate each probability, you divide each frequency by the total
prob_outcome
```

```
## myoutcome
##          2          4          5          6          7          8          9
## 0.06666667 0.13333333 0.06666667 0.03333333 0.23333333 0.13333333 0.10000000
##         10         11         12
## 0.10000000 0.06666667 0.06666667
```

```r
# replicate version (faster but little challenging)
myoutcome <- replicate(n=30, twodice_outcome())
outcome_table <- table(myoutcome)
n <- length(myoutcome)
prob_outcome <- outcome_table/n
prob_outcome
```

```
## myoutcome
##          3          4          5          6          7          8          9
## 0.03333333 0.06666667 0.13333333 0.03333333 0.20000000 0.23333333 0.10000000
##         10         11
## 0.10000000 0.10000000
```

```r
# how to ensure that it is proper probability?
sum(prob_outcome) # it needs to add up to 1
```

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

### Calculate the simulated expected value of the random variable.

```r
outcome_table * prob_outcome
```

```
## myoutcome
##          3          4          5          6          7          8          9
## 0.03333333 0.13333333 0.53333333 0.03333333 1.20000000 1.63333333 0.30000000
##         10         11
## 0.30000000 0.30000000
```

```
names(outcome_table) # we need this
```

```
## [1] "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11"
```

```
values <- as.numeric(names(outcome_table) )

# or
# values <- 2:12

sum(values * prob_outcome)
```

```
## [1] 7.5
```

```
mu_hat = sum(values * prob_outcome)

mu_hat
```

```
## [1] 7.5
```

### Calculate the simulated variance of the random variable.

```
sigma_2_hat <- sum( (values-mu_hat)^2 * prob_outcome  )
sigma_2_hat
```

```
## [1] 4.583333
```

### Make a function that has an input `n` and a list of output that gives you the simulated expected value and variance. (putting all together)

```
sim_fn <- function(n=1000){
 myoutcome = rep(0, n)

  for (i in 1:n){
    dice1 <- sample(1:6, 1)
    dice2 <- sample(1:6, 1)
    sum_of_two <- dice1 + dice2
    myoutcome[i] <- sum_of_two
  }

  outcome_table <- table(myoutcome)

  prob_outcome <- outcome_table/n

  values <- as.numeric(names(outcome_table) ) # it takes out the each column name and make it as numeri


  mu_hat = sum(values * prob_outcome)

  sigma_2_hat <- sum( (values-mu_hat)^2 * prob_outcome  )
  output <- list(mu = mu_hat, sigma2 = sigma_2_hat) # list output
  # output <- c(mu_hat, sigma_2_hat) # vector output
  return(output)
}
```

```r
sim_fn(100)
```

```
## $mu
## [1] 6.76
##
## $sigma2
## [1] 5.8224
```

```r
sim_fn(1000)
```

```
## $mu
## [1] 6.903
##
## $sigma2
## [1] 5.717591
```

```r
sim_fn(10000)
```

```
## $mu
## [1] 7.0052
##
## $sigma2
## [1] 5.860173
```

```r
sim_fn(100000)
```

```
## $mu
## [1] 7.01003
##
## $sigma2
## [1] 5.794289
```