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] "strawberry" "pear"       "orange"
```

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

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

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 "
```

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

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

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

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

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

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

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

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

```r
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] 4 6 3 3 4
```

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

```r
#
#
#
```

Function

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

```r
# Example: Function to calculate (a+b)^2
square <- function(a, b) {
  value = (a + b)^2  # Calculate the square of the sum of a and b
  return(value)      # Return the calculated value
}
square(1,2) # Output: 9
```

```
## [1] 9
```

```r
square(2,3) # Output: 25
```

```
## [1] 25
```

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

```r
# 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] 6 2 3 5 5
```

Let's roll one dice and show the output.

```r
#Outcome from those two dice

dice1 <- sample(1:6, 1)
dice2 <- sample(1:6, 1)

sum_of_two <- dice1 + dice2
sum_of_two
```

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

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] 8 7 3 6 9 3
```

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

```
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] 9
```

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

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

Calculate probability of each outcome.

```
# 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]  7  7  6  6  8  7  9  7  7  9  5  7  4  5  9 10  7  3  3  9  6  9  6  6 12
## [26]  4  6  8 11  7
```

```
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
##          3          4          5          6          7          8          9
## 0.06666667 0.06666667 0.06666667 0.20000000 0.26666667 0.06666667 0.16666667
##         10         11         12
## 0.03333333 0.03333333 0.03333333
```

```
# 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
## 0.16666667 0.10000000 0.06666667 0.13333333 0.10000000 0.13333333 0.13333333
```

```
##         11         12
## 0.10000000 0.06666667
```

```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        11
## 0.8333333 0.3000000 0.1333333 0.5333333 0.3000000 0.5333333 0.5333333 0.3000000
##        12
## 0.1333333
```

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

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

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

# or
# values <- 2:12

sum(values * prob_outcome)
```

```
## [1] 6.9
```

```r
mu_hat = sum(values * prob_outcome)

mu_hat
```

```
## [1] 6.9
```

Calculate the simulated variance of the random variable.

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

```
## [1] 7.89
```

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)

```r
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
  }
```

```r
  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)
}


sim_fn(100)
```

```
## $mu
## [1] 6.87
##
## $sigma2
## [1] 5.4731
```

```r
sim_fn(1000)
```

```
## $mu
## [1] 6.926
##
## $sigma2
## [1] 5.996524
```

```r
sim_fn(10000)
```

```
## $mu
## [1] 6.9577
##
## $sigma2
## [1] 5.770711
```

```r
sim_fn(100000)
```

```
## $mu
## [1] 6.9978
##
## $sigma2
## [1] 5.801195
```