# Lab 1: Introduction to R

San Diego State University - STAT550

Aiden Jajo

**"Task 1: A Basic Simulation Event"** Much of our applied probability computing work in this class will be simulating events. This means that we generate an event at random. The R function for simulating random numbers is sample; check out the help screen.

**Code set-up** Let us try simulating a die roll: The parameter replace = TRUE is important here as we are rolling the die over and over again, not drawing marbles out of a bag. Here is how to roll a 6-sided die five times in R, and then compute the average of the rolls. Try running it!

```
x = 1:6  # sides of the die
roll = sample(x, 5, replace = TRUE)  # tell R how many sides (x) and how many rolls (5)
mean(roll) # average of the 5 rolls
```

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

## The problem

A tetrahedron die is a four-sided die with labels {1, 2, 3, 4}. Have R make 10 rolls of the tetrahedron die and compute the average. (Can think of this as rolling 10 different tetrahedron dice as well.) Keep the output in this RMarkdown file for grading purposes.

```
# Define 4-sided die
x = 1:4

# Roll the tetrahedron die 10 times
roll = sample(x, 10, replace = TRUE)

# Display results
print(roll)
```

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

```
print(mean(roll))
```

```
## [1] 2.5
```

## Question:

What value do you expect to get for the average of the 10 rolls?

*I would expect around 2.5 because there is a one fourth chance of each side.*

## Task 2: Playing with for-loops

For-loops are central to the simulation studies we will be performing in this class. In these experiments, simulation tasks are repeated over and over again. The for-loop can easily perform this replication for us. The trick is appropriately storing your results for analysis. The syntax for a for-loop in R is for(var in seq){task}, read "for a given variable in a specified sequence." The for-loop steps that variable through the sequence and performs the task each time.

### Code set-up

Let us apply a for-loop for simulating a 6-sided die roll. That is, repeat 1000 times the experiment of rolling a 6-sided die five times and computing the average.

```r
S = 1000 # number of simulation experiments performed

# store results in a (1000-dimensional) vector called rolls.avgs
rolls.avgs = vector(length=S)  # setting up the variable rolls.avgs to store the average roll for each

# this for-loop steps the variable simnum through the sequence 1 to 1000,
# repeating 1000 times the die rolling tasks inside the curly brackets {...}.
for(simnum in 1:S){
  # Use our die rolling code from Task 1!
  x = 1:6 # sides of the die
  roll = sample(x, 5, replace = TRUE)  # simulate a die roll
  rolls.avgs[simnum] = mean(roll) # store the average roll
}

# take a look at the first 6 simulation results
head(rolls.avgs)
```

```
## [1] 4.4 3.2 4.2 4.0 4.8 4.4
```

```r
# compute the mean of the 1000 experiments
mean(rolls.avgs)
```

```
## [1] 3.5082
```

### The problem

Repeat 1000 times the experiment you performed in Task 1, that is rolling a tetrahedron die 10 times and computing the average. Report the average and standard deviation of the 1000 experiments. The standard deviation function in R is sd(x).

```r
S = 1000

# Initialize vector to store average from each experiment
rolls.avgs = vector(length=S)

# Run 1000 simulation experiments
for(simnum in 1:S) {
  x = 1:4  # tetrahedron die
  roll = sample(x, 10, replace = TRUE)  # 10 rolls per experiment
  rolls.avgs[simnum] = mean(roll)  # store average
}

# Report mean and standard deviation
print(paste("Mean of 1000 experiments:", mean(rolls.avgs)))
```

```
## [1] "Mean of 1000 experiments: 2.4707"
```

```
print(paste("Standard deviation of 1000 experiments:", sd(rolls.avgs)))
```

```
## [1] "Standard deviation of 1000 experiments: 0.345848407718065"
```

### Questions:

- Is the mean closer to the value you would expect than the average you had in Task 1? Why?

*Yes, the mean from 1000 experiments is closer to the expected value than the average from Task 1. Completing 1000 experiments cuts down on the impact of random variations. This gets us closer to the expected value.*

- How do you interpret the standard deviation in this problem?

*The standard deviation represents the variability in the average rolls from all 1000 experiments. It shows how the average of 10 rolls generally varies and a smaller standard deviation means the averages are more consistent.*

## Task 3: Presenting tables in RMarkdown

Let us present a table of our die rolls. We will use xtable and pander R packages. Make sure to install the pander package prior to running the code chunk. In this task, we will also try the replicate() function in R to replace the for-loop. Have R make 5 rolls of the tetrahedron die and repeat that 4 times. Present the results in a table.

## R Markdown

The exact code is provided for you below. In this way you can cut-and-paste this code for table-making in future labs. Three parameters were added to the code chunk: The echo = FALSE parameter was addedto prevent printing of the R code that generated the table. The results='asis' parameter was added to have R present the results as is for the table generation. The warning=FALSE parameter suppresses warning messages from R that are often presented when loading packages.

As an aside, a fourth common parameter is include=FALSE, which prevents R from printing output when running the code chunk.

```
# we will create a table using xtable and pander
library(knitr)
library(xtable)
library(pander)
```

```
# replicate() function: repeat an operation multiple times
# Here we roll the die 5 times, and replicate that 4 times
R = replicate(4, sample(1:4, 5, replace = TRUE))

# Create formatted table with caption
table = xtable(R, caption="Replicate 5 rolls of a tetrahedron die four times", align = "|2|rrrr|")
```

```
## Warning in .alignStringToVector(value): Nonstandard alignments in align string
```

```
names(table) <- c('Replicate 1', 'Replicate 2', 'Replicate 3', 'Replicate 4')

# Display table
pander(table)
```

Table 1: Replicate 5 rolls of a tetrahedron die four times

| Replicate 1 | Replicate 2 | Replicate 3 | Replicate 4 |
|---|---|---|---|
| 4 | 3 | 4 | 1 |
| 2 | 2 | 2 | 3 |
| 2 | 2 | 4 | 4 |
| 1 | 2 | 4 | 1 |
| 4 | 2 | 3 | 1 |

### Question:

What do you observe across the replicates?

*Across the replicates, I observed different random outcomes. Each replicate shows a different sequence of numbers between 1 and 4. This demonstrates the random outcome possibilities of die rolling, with all values varying but staying within the 1-4 range.*

## Task 4: Presenting graphs in RMarkdown

Graphs are easy to display in an RMarkdown file.

### Code set-up

Let us draw a histogram of our 1000 die rolls from earlier.

```
S = 1000 # number of simulation experiments performed

# store results in a (1000-dimensional) vector called rolls.avgs
rolls.avgs = vector(length=S) # setting up the variable rolls.avgs to store the average roll for each e

# this for-loop steps the variable simnum through the sequence 1 to 1000,
# repeating 1000 times the die rolling tasks inside the curly brackets {...}.
for(simnum in 1:S){
  # Use our die rolling code from Task 1!
  x = 1:6 # sides of the die
```

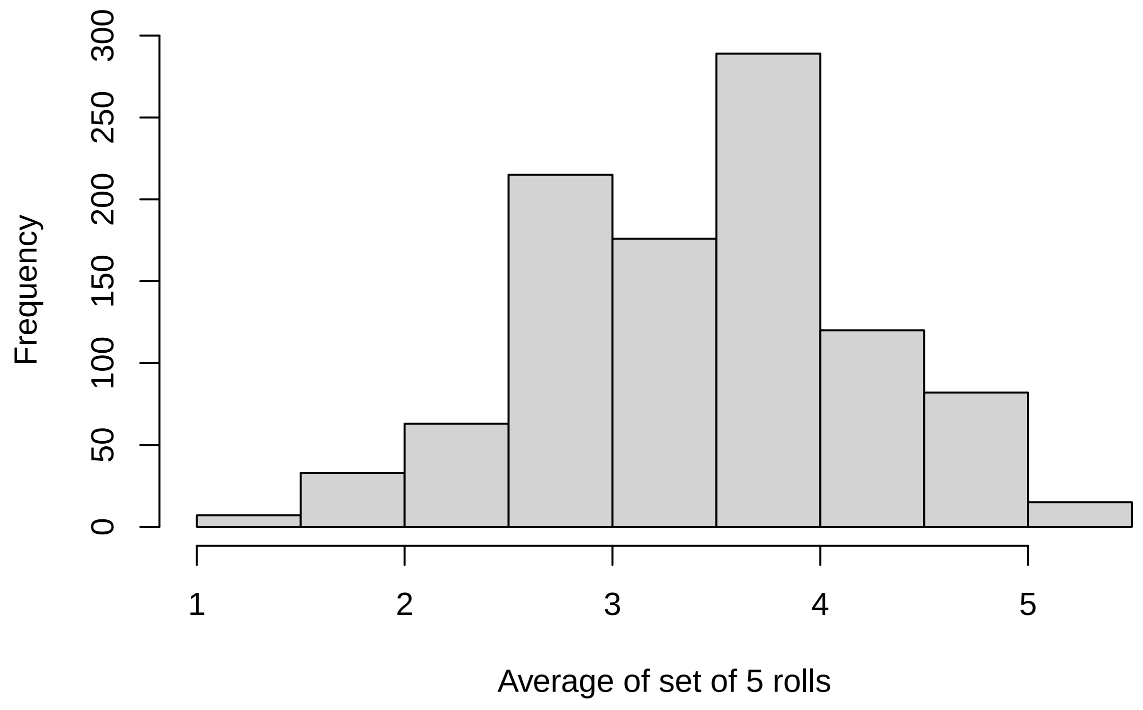
```r
  roll = sample(x, 5, replace = TRUE) # simulate a die roll
  rolls.avgs[simnum] = mean(roll) # store the average roll
}

# take a look at the first 6 simulation results
head(rolls.avgs)
```

```
## [1] 4.4 3.6 4.0 4.0 3.4 3.2
```

```r
# compute the mean of the 1000 experiments
mean(rolls.avgs)
```

```
## [1] 3.486
```

```r
# create histogram of results
hist(rolls.avgs, main="", xlab="Average of set of 5 rolls")
```



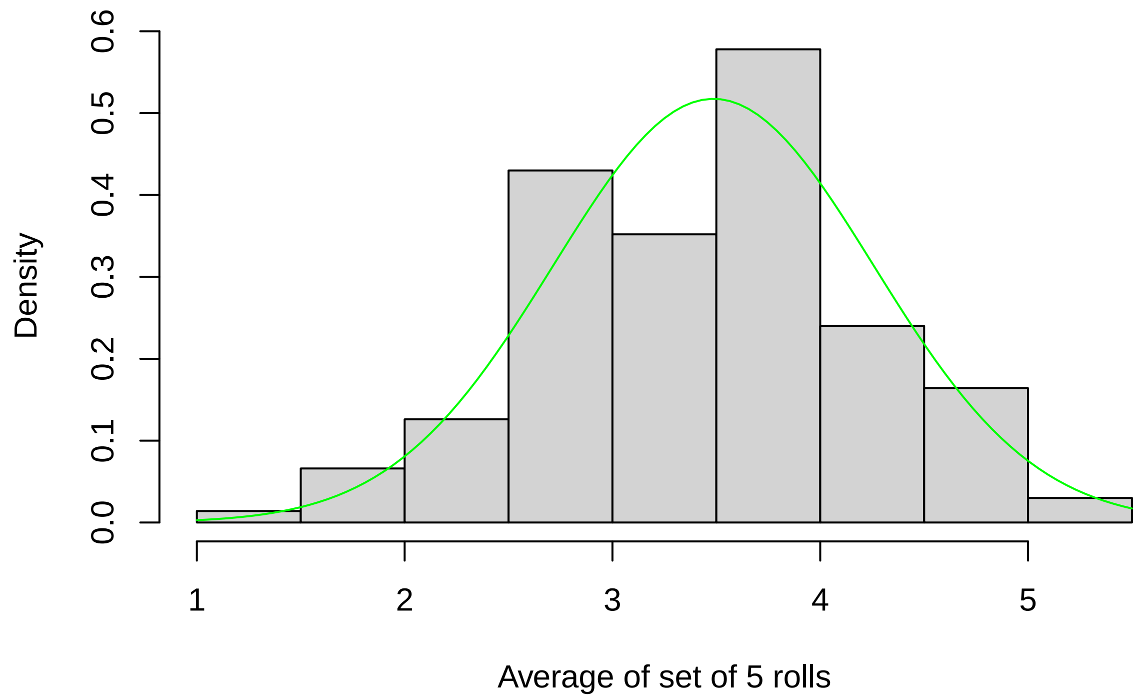
## The problem

Let us add a normal approximation (bell curve) to the histogram. We will cover the normal distribution later in the course. But hopefully you recall it from your Statistics course! To add a density curve to the plot, need to change the y-axis to a 'density' scale. This is done by setting the parameter prob = TRUE. The curve function addes a curve to the plot. We will use a normal distribution with mean and standard deviation set at the values obtained in Task 2. Here is the code

```
#hist(rolls.avgs, prob = T, main="", xlab="Average of set of 5 rolls") # histogram
#curve(dnorm(x, mean=mean(rolls.avgs), sd=sd(rolls.avgs)), add=TRUE, col="green") # normal approximatio
```

Add these to the code chunk to present a histogram with a normal approximation

```
# Create histogram with density scale (prob = T)
hist(rolls.avgs, prob = T, main="", xlab="Average of set of 5 rolls")

# Overlay normal distribution curve with same mean and SD
curve(dnorm(x, mean=mean(rolls.avgs), sd=sd(rolls.avgs)), add=TRUE, col="green")
```



### Questions:

- Interpret the histogram–shape, skew, spread, center.

*This histogram shows a rough bell-shaped distribution around 3.5. The shape is also approximately symmetric with very little to no skew. The distribution has a moderate spread. Most values fall between 2 and 5 with the center being about 3.5.*

- Does this follow what you would expect to see?

*Yes, this follows exactly what I would expect to see. The distribution of sample means tends toward a normal distribution, even when the underlying data is not normally distributed according to the Central Limit Theorem. While the number of samples increases, the averages become more bell-shaped and centered around the anticipated value.*

## Task 5: Boolean expressions

Another useful task is making logical statements in R.

## Code set-up

Let us first make 10 rolls of a die and see how often a 6 is rolled.

```r
x = 1:6  # 6-sided die
rolls = sample(x, 10, replace = TRUE)  # roll the die ten times

# Boolean expression: how often is the roll EXACTLY 6, use double-equals sign
sum(rolls == 6)
```

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

Now repeat 1000 times the experiment of rolling a die 10 times as in Task 2. We will see how many times a six occurs at least once out of ten rolls across all these experiments. The code for this counting exercise is sum(roll==6)>0 since a "success" is an experiment where the total number of sixes showing on ten rolls is more than zero!

```r
six = 0 # start a counter for number of times at least one six shows in 10 rolls
S = 1000 # number of experiments

# for-loop to repeat die rolling experiment 1000 times
for(simnum in 1:S){
  x = 1:6 # 6-sided die
  roll = sample(x, 10, replace = TRUE) # roll the die ten times

  # two ways to count: with an if-then statement, or more elegantly with a Boolean computation
  #if(sum(roll==6) > 0){six = six + 1}  # if-then statement
  six = six + (sum(roll==6)>0)  # Boolean computation: add one to the counter only if at least one 6 sh
}

# Display count of experiments with at least one six
six
```

```
## [1] 835
```

## The problems

First problem How often in 5 rolls of a tetrahedron die is a two rolled?

```r
x = 1:4  # tetrahedron die
rolls = sample(x, 5, replace = TRUE)  # 5 rolls

# Count how many times a 2 appears
print(sum(rolls == 2))
```

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

## Questions:

- Run the code multiple times. What values do you get?

*After running the code multiple times, I got values like 0, 1, 2, or 3. These values represent the number of twos in 5 rolls.*

- Are the values different? Is that what you expect?

*Yes, I would say the values are different but expected since each run is a new random experiment. The random variation means we will get different outcomes every time.*

Second problem This is heading towards a probability calculation. Roll the tetrahedron die 5 times and repeat this experiment 1000 times as in Task 2. Report the proportion of 1000 simulations where a two occurred. (This derives from Dobrow problem 1.44: Probability of rolling at least one 2 in five rolls of a tetrahedron die.) Dobrow problem 1.30: Exact answer is 0.7627

```r
two_count = 0  # counter for experiments with at least one two
S = 1000  # number of experiments

# Run 1000 simulation experiments
for(simnum in 1:S) {
  x = 1:4  # tetrahedron die
  roll = sample(x, 5, replace = TRUE)  # 5 rolls per experiment

  # If at least one 2 appears, increment counter
  if(sum(roll == 2) > 0) {
    two_count = two_count + 1
  }
}

# Calculate proportion
proportion = two_count / S
print(paste("Proportion with at least one two:", proportion))
```

```
## [1] "Proportion with at least one two: 0.771"
```

## Questions:

- Is the value you get close to the truth (0.7627)?

*Yes, the value I get is close to the truth. The slight difference is because of the random variation in the simulation.*

- How can we modify the simulation experiment to get a value even close to the truth?

*In order to get closer to the true value, we could raise the number of experiments conducted from 1000 to 10000 or more. Conducting more experiments would reduce the sampling error and give a more precise outcome.*