P_M1_1

September 6, 2021

This assignment will be reviewed by peers based upon a given rubric. Make sure to keep your answers clear and concise while demonstrating an understanding of the material. Be sure to give all requested information in markdown cells. It is recommended to utilize Latex.

0.0.1 Problem 1

The Birthday Problem: This is a classic problem that has a nonintuitive answer. Suppose there are N students in a room.

Part a) What is the probability that at least two of them have the same birthday (month and day)? (Assume that each day is equally likely to be a student's birthday and that there are no sets of twins.)

Note: Jupyter has two types of cells: Programming and Markdown. Programming is where you will create and run R code. The Markdown cells are where you will type out expalantions and mathematical expressions. Here is a document on Markdown some basic markdwon syntax. Also feel free to look at the underlying markdown of any of the provided cells to see how we use markdown.

P({At least two people in a group of N people have same birthday}) = 1 - 365! / ((365-N)! * (365^N))

Part b) How large must N be so that the probability that at least two of them have the same birthday is at least 1/2?

[122]:
```
# N poplation
N = 50
# Empty vector to efficiently store values
probs <-numeric(N)

for (i in 1:N){
        # probability of a match in a group of k people is 1- 365! / 365-k! *
 ↪3665^k
        probs[i]  <- 1- prod(1 - (0:(i - 1))/365)}

print('The number of people needed for a 50% match probability is:')
```

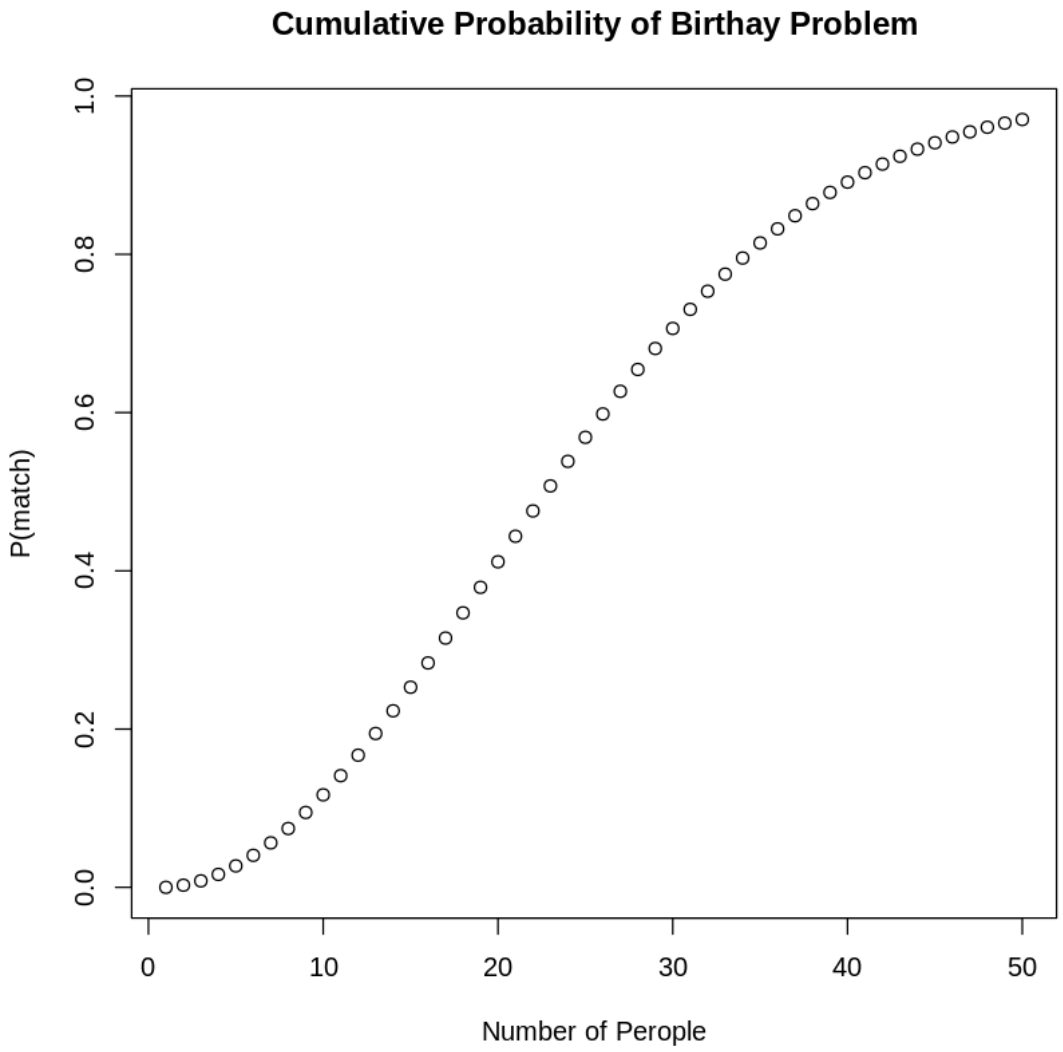
```
# Find the first number in the probability vector that is greater than 50% or 0.
↪5
min_people_needed = which(probs> 0.5)[1]

print(min_people_needed)
```

```
[1] "The number of people needed for a 50% match probability is:"
[1] 23
```

The minimum number of people for the sample to have a 50% probability of at least one birthday match, given the constraints of the problem, is 23.

Part c) Plot the number of students on the x-axis versus the probability that at least two of them have the same birthday on the y-axis.

[121]:
```
plot(probs, main = 'Cumulative Probability of Birthay Problem',␣
↪ylab='P(match)', xlab='Number of Perople')
```



Thought Question (Ungraded) Thought question (Ungraded): Would you be surprised if there were 100 students in the room and no two of them had the same birthday? What would that tell you about that set of students?

Yes, that would be surprising. Based on our probability distribution 100 randomly selected people with equally likely birthdays is exceedingly unlikely (<0.0001% probability) to not have at least one matching pair of birthdays. If the set of students did not have a matching brithday pair, it would suggest the set of students was carefully chosen for that condition somehow.

1 Problem 2

One of the most beneficial aspects of R, when it comes to probability, is that it allows us to simulate data and random events. In the following problem, you are going to become familiar with these simulation functions and techniques.

Part a)

Let X be a random variable for the number rolled on a fair, six-sided die. How would we go about simulating X?

Start by creating a list of numbers [1, 6]. Then use the `sample()` function with our list of numbers to simulate **a single** roll of the die, as in simulate X. We would recommend looking at the documentation for `sample()`, found here, or by executing `?sample` in a Jupyter cell.

[66]:
```
die_numbers = c(1,2,3,4,5,6)

# simulate single roll of the die
sample(die_numbers, 1)
```

2

Part b)

In our initial problem, we said that X comes from a fair die, meaning each value is equally likely to be rolled. Because our die has 6 sides, each side should appear about $1/6^{th}$ of the time. How would we confirm that our simulation is fair?

What if we generate multiple instances of X? That way, we could compare if the simulated probabilities match the theoretical probabilities (i.e. are all 1/6).

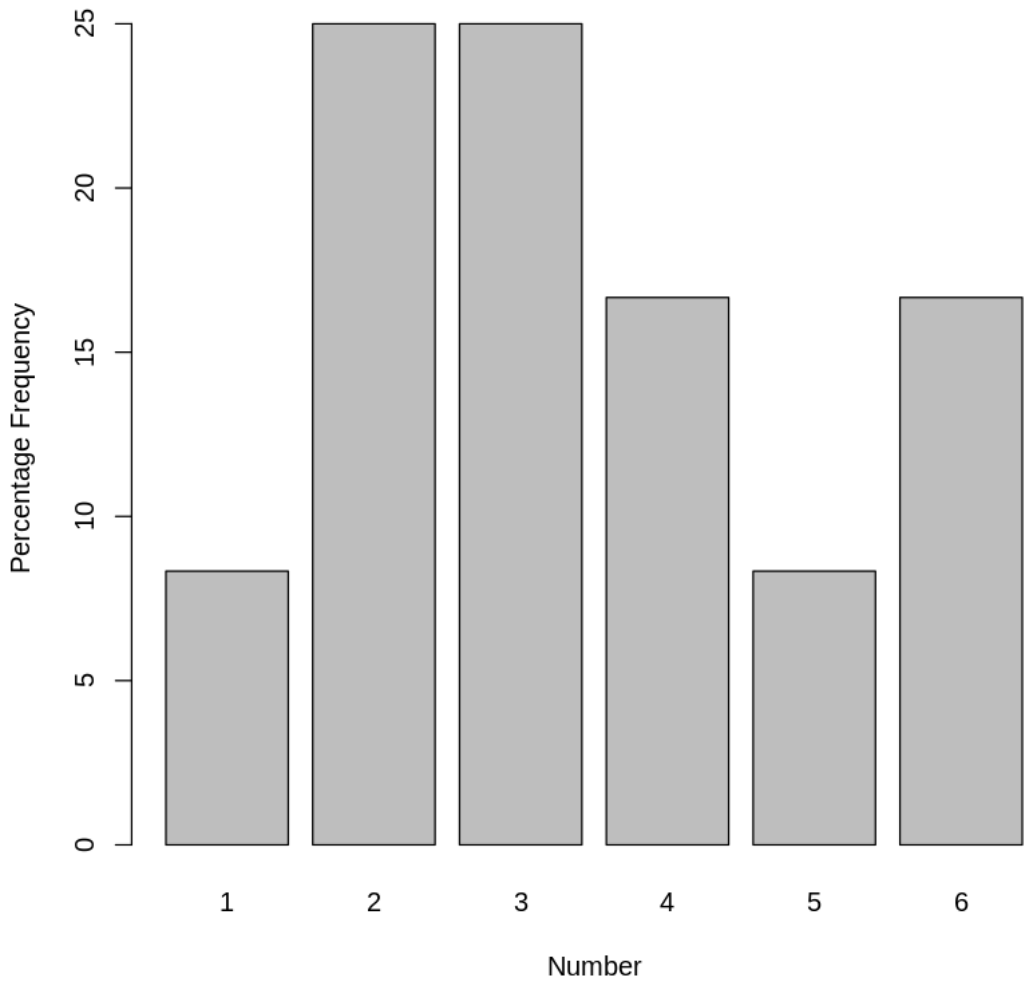
Generate 12 instances of X and calculate the proportion of occurances for each face. Do your simulated results appear to come from a fair die? Now generate 120 instances of X and look at the proportion of each face. What do you notice?

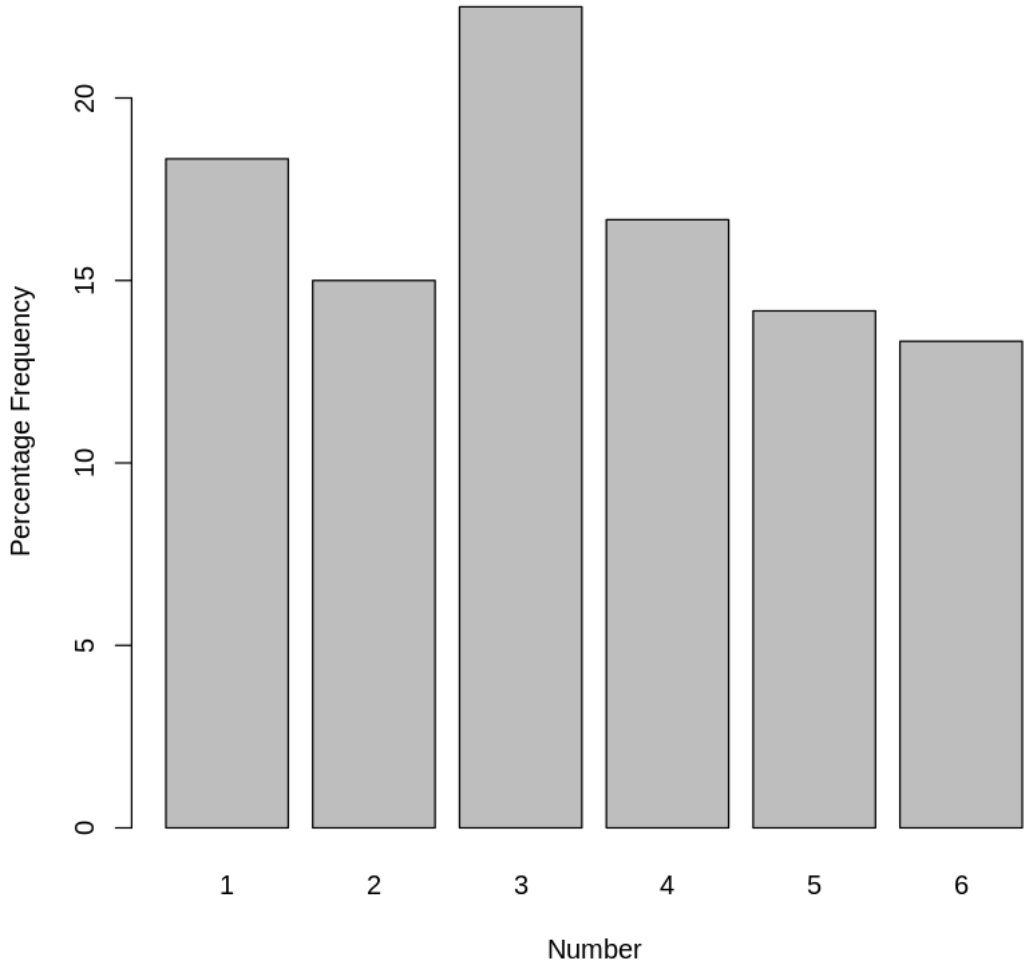
Note: Each time you run your simulations, you will get different values. If you want to guarantee that your simulation will result in the same values each time, use the `set.seed()` function. This function will allow your simulations to be reproducable.

[48]:
```
library("plyr")
set.seed(112358)

# Your Code Here
roll_n_times <- function(n){
    results = sample(die_numbers, n, replace=TRUE)
    # convert results to a frequency table of percentages
    results <- (count(results)/n)*100
    print(results)
    # create barplot of results
    barplot(results$freq, names.arg = c(1,2,3,4,5,6),xlab='Number',␣
 ↪ylab='Percentage Frequency')
```

```
}

print("frequencies here are stored in results$freq and visualized in the␣
 ↪barplots")
roll_n_times(12)
roll_n_times(120)
```

```
[1] "frequencies here are stored in results$freq and visualized in the barplots"
          x      freq
1  8.333333  8.333333
2 16.666667 25.000000
3 25.000000 25.000000
4 33.333333 16.666667
5 41.666667  8.333333
6 50.000000 16.666667
          x     freq
1 0.8333333 18.33333
2 1.6666667 15.00000
3 2.5000000 22.50000
4 3.3333333 16.66667
5 4.1666667 14.16667
6 5.0000000 13.33333
```


Percentage Frequency
0
5
10
15
20
25
1
2
3
4
5
6
Number



Random processes may appear to be skewed or non-random with small sample sizes. But as the number of samples becomes quite large, the "true" distribution tends to become clearer. In the parameters of this problem, the distribution appears to be skewed with just 12 rolls with 1 and 5 seemingly much less likely to be rolled than 2 and 3. But after 120 rolls, it becomes more apparent that each number is equally likely to occur.

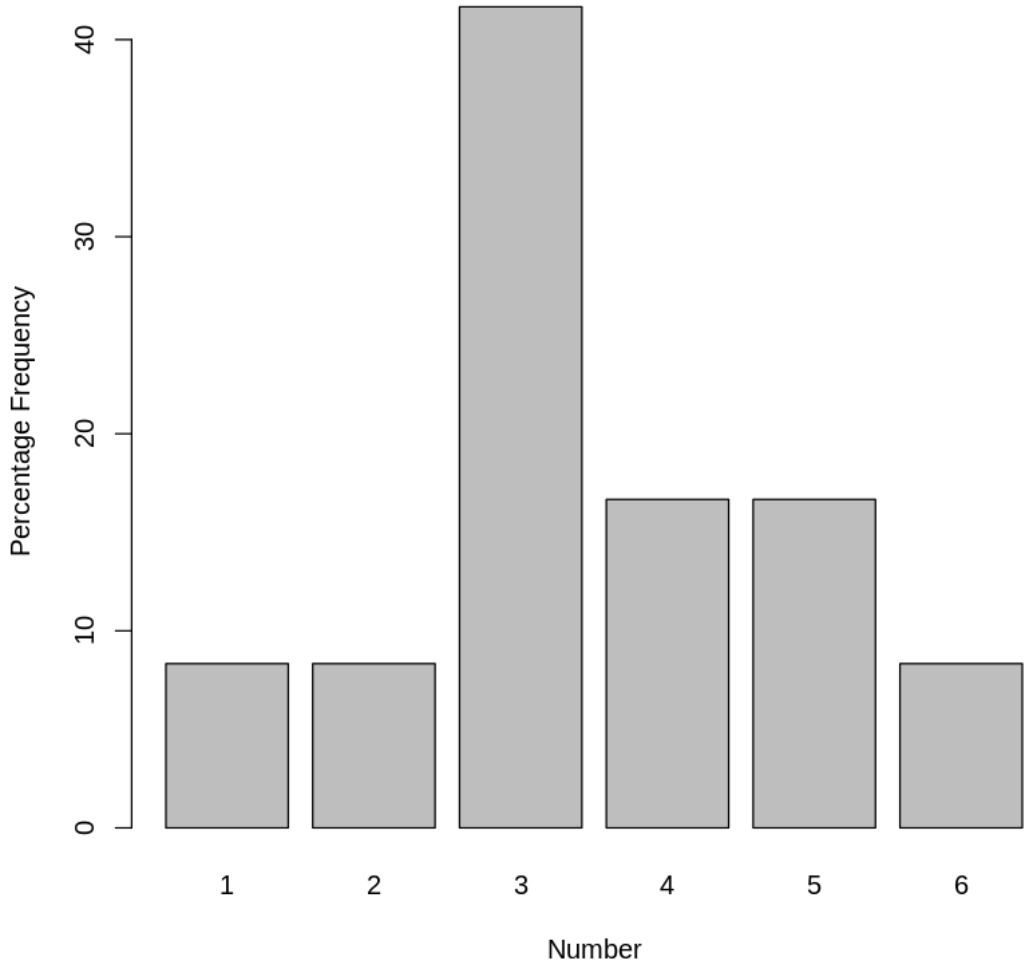
Part c)

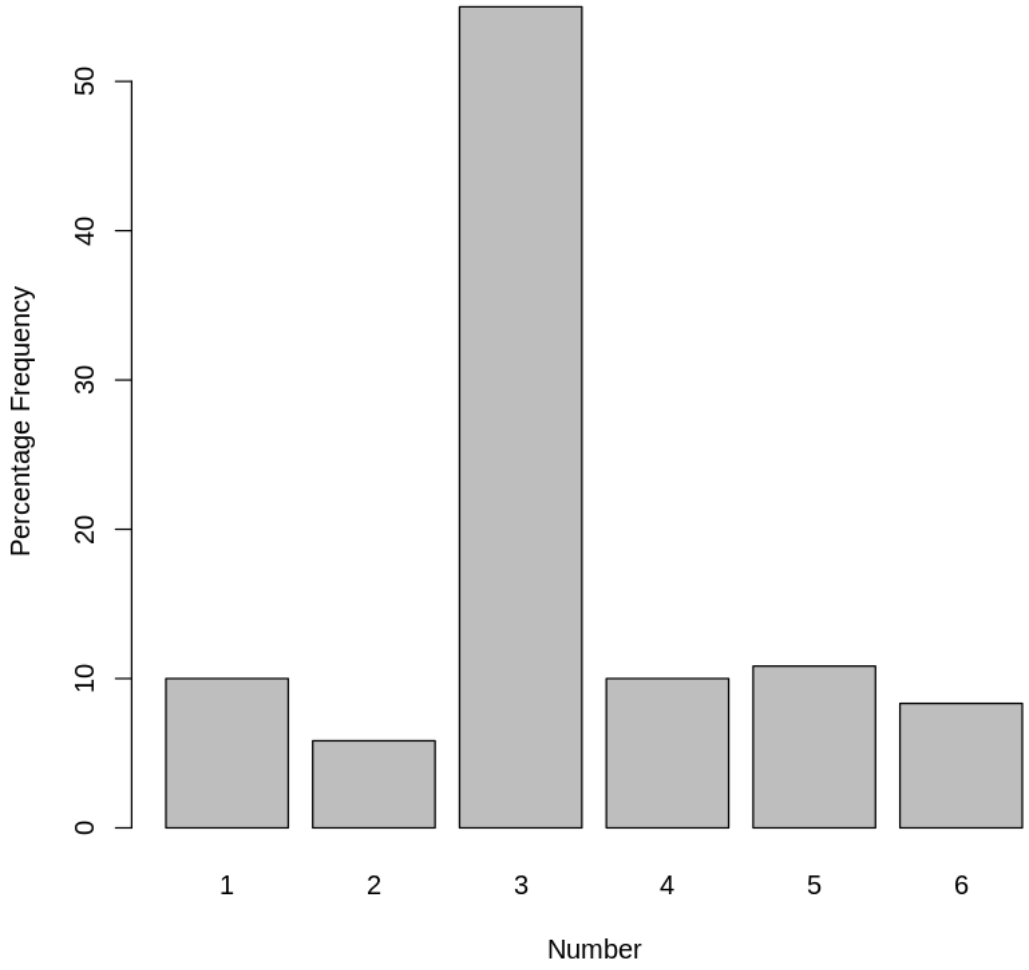
What if our die is not fair? How would we simulate that?

Let's assume that Y comes from an unfair six-sided die, where $P(Y = 3) = 1/2$ and all other face values have an equal probability of occuring. Use the `sample()` function to simulate this situation. Then display the proportion of each face value, to confirm that the faces occur with the desired probabilities. Make sure that n is large enough to be confident in your answer.

[124]:
```r
set.seed(24)

# Your Code Here
roll_n_times_weighted <- function(n){
    # vector of probability weights
    prob_vector = c(0.1, 0.1, 0.5, 0.1, 0.1, 0.1)
    results = sample(die_numbers, n, replace=TRUE, prob= prob_vector)
    results <- (count(results)/n)*100
    print(results)
    barplot(results$freq, names.arg = c(1,2,3,4,5,6), xlab='Number',␣
 ↪ylab='Percentage Frequency')
}

roll_n_times_weighted(12)
roll_n_times_weighted(120)
```

```
          x      freq
1  8.333333  8.333333
2 16.666667  8.333333
3 25.000000 41.666667
4 33.333333 16.666667
5 41.666667 16.666667
6 50.000000  8.333333
          x      freq
1 0.8333333 10.000000
2 1.6666667  5.833333
3 2.5000000 55.000000
4 3.3333333 10.000000
5 4.1666667 10.833333
6 5.0000000  8.333333
```


Percentage Frequency
0
10
20
30
40
1
2
3
4
5
6
Number



The sample() function allows you to specify a probability vector to weight each possibile outcome. To simulate the problem parameters you can create a vector where 3 has a 50% probability of being rolled while all other numbers have an equal 10% chance of being rolled (all adding up to 100%). With a sufficiently large sample size it becomes apparent in the randomly generated data that the number three is favored and the die is not fair.

[ ]: