P_M1_1

August 30, 2022

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, that there are no sets of twins, and that there are 365 days in the year. Do not include leap years).

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.

$$\begin{aligned} P(\text{At least two have same birthday}) &= 1 - P(\text{No one has the same bday}) \\ &= 1 - (\frac{364}{365})^{C(N,2)} \end{aligned}$$

[29]:
```
### Library import ###
library(tidyverse)

### Helper Functions ###
combinations = function(n, k){
    #' n-choose-k
    #'
    #' @param n The total number of objects in the set.
    #' @param k The number chosen from the set.
    ## Notes: factorial may hit a limit at ~factorial(172)

    ## This implementation decreases R from hitting the limit and returns␣
↪infinity
    result = 1  # multiplicative identity
```

```
    for (i in 1:k){
        result = result * (n - (i-1))
    }
    result = result / factorial(k)
    return(result)
}

permutations = function(n, k){
    #' n-permute-k
    #'
    #' @param n The total number of objects in the set.
    #' @param k The number chosen to permute from the set.

    return( factorial(n) / factorial(n-k) )
}

bose_einstein = function(n, k){
    #' Bose-Einstein
    return(factorial(n+k-1) / (factorial(n+k-1-k)*factorial(k)))
}
```

[45]:

```
### Variables ###
N = 23  # Number of students
simulation_rep = 300  # Simulation repetitions




### Method #1 - Using the reciprocal probability ###
## Notes: Easier to find the reciprocal, which is proba of no-one having the␣
↪same bday.
## Notes: Remember that we are not just comparing with oneself, each student in␣
↪the
##        group needs to compare with the rest
prob_have_same_bday = function(num_of_students){

    ## Calculations
    prob_noOne_same_bday = (364/365)**(combinations(num_of_students, 2))

    ## Finding the reciprocal
    return(1-prob_noOne_same_bday)
}

## Finding the reciprocal
answer_1 = prob_have_same_bday(N)
print(sprintf("answer_1_deterministic (prob of atLeast two have same bday): %.
↪15f", answer_1))
```

```r
### Method #2 - Simulation ###
sim_prob_have_same_bday = function(num_rep, N){

    ## Placeholder
    num_of_rep_with_overlap_bdays = 0
    sim_rep_prob = rep(NA, num_rep)

    ## Simulation
    for (i in 1:num_rep){
        #sim_rep = vector(mode="list", length=N)  # Vector to store simulation
        sim_rep = rep(NA, N)
        for (j in 1:N){
            sim_rep[j] = sample(1:365, 1)
        }

        ## Convert to tibble
        df_freq = as_tibble(table(sim_rep)) %>% setNames(c("bday", "freq"))
        ## Find days with more than one bdays
        df_overlap_bday = df_freq %>% filter(freq > 1)
        ## Find number of days with more than one bdays
        num_of_overlap_bdays = dim(df_overlap_bday)[1]

        ## If rep have at least one overlap bday day
        if (num_of_overlap_bdays >= 1) {
            num_of_rep_with_overlap_bdays = num_of_rep_with_overlap_bdays + 1
        }
    }
    prob_with_atLeast_one_sameBday = num_of_rep_with_overlap_bdays / num_rep
    return(prob_with_atLeast_one_sameBday)
}
answer_2 = sim_prob_have_same_bday(simulation_rep, N)
print(sprintf("answer_2_simulation (prob of atLeast two have same bday): %.
↪15f", answer_2))
```

```
[1] "answer_1_deterministic (prob of atLeast two have same bday):
0.500477154036581"
[1] "answer_2_simulation (prob of atLeast two have same bday):
0.536666666666667"
```

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?

With $N = 23$, the probability of at least two of the students have the same bday is 0.5004771540, which is greater than $\frac{1}{2}$.

[36]:
```
## Finding the point where probability is greater than 1/2
for (i in 2:365){
    prob = prob_have_same_bday(i)
    ## Printing only a subset
    if ((prob > 0.4) & (prob < 0.6)) {
        print(sprintf("N=%d: probability=%.10f", i, prob))
    }
}
```

```
[1] "N=20: probability=0.4062294595"
[1] "N=21: probability=0.4379317800"
[1] "N=22: probability=0.4693991596"
[1] "N=23: probability=0.5004771540"
[1] "N=24: probability=0.5310232667"
[1] "N=25: probability=0.5609077642"
[1] "N=26: probability=0.5900142731"
```

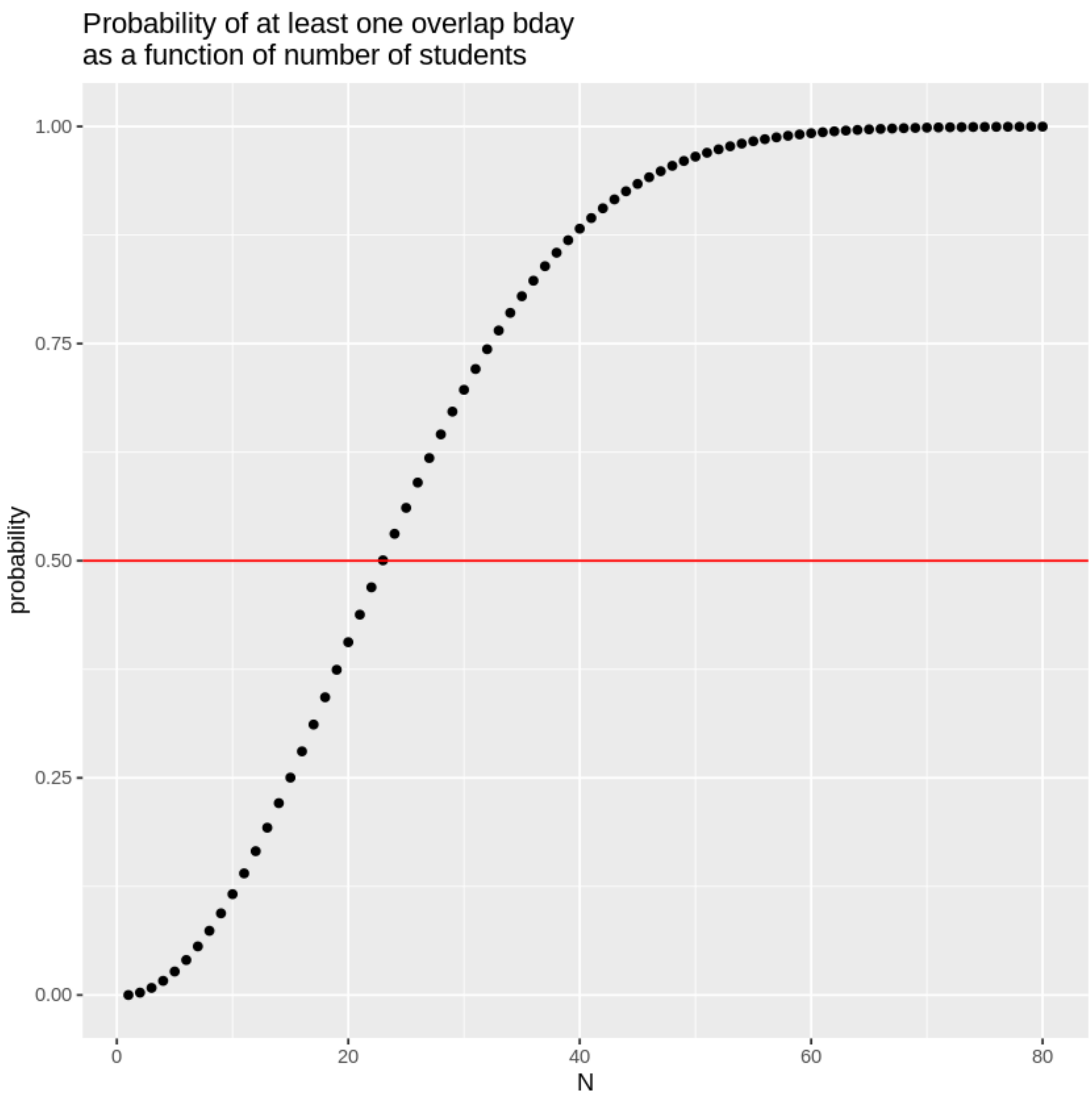
Part c) Plot the number of students on the x-axis versus the probability that at least two of them have the same birthday on the y-axis.

See the plot below

[42]:
```
## Library
library(ggplot2)

## Create probability vector
probability_list = rep(NA, 365)
for (i in 1:365){
    probability_list[i] = prob_have_same_bday(i)
}

## Create a tibble using vectors
probability_table = tibble(N=1:365, probability=probability_list)

## Filter the tibble
# print(probability_table %>% filter(probability > 0.5))

## Plot
ggplot(probability_table, aes(x=N, y=probability)) +
    ggtitle("Probability of at least one overlap bday\nas a function of number␣
 ↪of students") +
    geom_point() +
    geom_hline(yintercept=0.5, color="red") +
    xlim(1, 80)
```

```
Warning message:
```

"Removed 285 rows containing missing values (geom_point)."



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, I would be really surprised as the probability of that is very small, less than 1.5 in a million.

[44]:
```
1-prob_have_same_bday(100)
```

1.26523152166325e-06

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.

[48]:
```
# Your Code Here
# ?sample

rv_X = sample(x=1:6, size=1)
print(rv_X)
```

```
[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.

[69]:
```
set.seed(112358)
# Your Code Here
### Helper function ###
row_dice = function(num_instances, fair=TRUE, prob=rep(1/6, 6)){
    all_instances = rep(NA, num_instances)

    if (fair == TRUE){
        for (i in 1:num_instances){
            all_instances[i] = sample(x=1:6, size=1)
        }
    } else if (fair == FALSE){
```

```r
        if (identical(prob,rep(1/6, 6))){
            print("Probability of instances is not adjusted.")
        } else {
            for (i in 1:num_instances){
            all_instances[i] = sample(x=1:6, size=1, prob=prob)
            }
            print("This is a trickster dice! Something is not right with it!")
        }

    } else {print("Fairness of the dice is not indicated.")}

    ## Count the instances
    df_counts = as.tibble(table(all_instances)) %>%
                    setNames(c("dice_face", "freq")) %>%
                    mutate(freq_percentage = freq/num_instances * 100)
    print(sprintf("%d dice rolls and its frequencies: ", num_instances))
    print(df_counts)
}


### 12 instances ###
row_dice(12)


### 120 instances ###
row_dice(120)


### 1200 instances ###
row_dice(1200)
```

```
[1] "12 dice rolls and its frequencies: "
# A tibble: 6 x 3
  dice_face  freq freq_percentage
  <chr>     <int>
<dbl>
1 1             1            8.33
2 2             3           25
3 3             3           25
4 4             2           16.7
5 5             1            8.33
6 6             2           16.7
[1] "120 dice rolls and its frequencies: "
# A tibble: 6 x 3
  dice_face  freq freq_percentage
  <chr>     <int>
<dbl>
```

```
1 1            22            18.3
2 2            18            15
3 3            27            22.5
4 4            20            16.7
5 5            17            14.2
6 6            16            13.3
[1] "1200 dice rolls and its frequencies: "
# A tibble: 6 x 3
  dice_face  freq freq_percentage
  <chr>     <int>
<dbl>
1 1           203            16.9
2 2           211            17.6
3 3           181            15.1
4 4           191            15.9
5 5           224            18.7
6 6           190            15.8
```

As the number of instances increases from 12, to 120, to 1200, the frequency resembles more of 1/6.

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.

[70]:
```r
# Your Code Here
row_dice(3000, fair=FALSE, prob=c(1/10, 1/10, 1/2, 1/10, 1/10, 1/10))
```

```
[1] "This is a trickster dice! Something is not right with it!"
[1] "3000 dice rolls and its frequencies: "
# A tibble: 6 x 3
  dice_face  freq freq_percentage
  <chr>     <int>
<dbl>
1 1           299             9.97
2 2           297             9.9
3 3          1521            50.7
4 4           300            10
5 5           299             9.97
6 6           284             9.47
```

As we can see, here $P(Y = 3) = 50.7$, approximating the probability of 1/2 that we specified. This is a trickster dice!

[ ]:
```

```