HW1

Gabriel Nespoli / Fernando Crema / Mauricio Fadel Argerich

October 9, 2016

Contents

Explanation of solution

Structure

We submitted the whole homework in one .Rmd. In the directory, you'll find 2 folders, 1 .Rmd and the tsv file:

1. **images:** Images for the generation of the .html.
2. **src:** Various independent files which helped to do the simulation and create the images used in this .Rmd.
3. **G09.Rmd:** This file
4. **you.tsv:** The tab separated values used in part II.

Repository

To fork repository or download a pdf version of this .Rmd go to stat4DS-hw1

Part I: R Syntax & Functions

a. Explain errors or non-erroneous result.

The command creates a vector with 4 elements that are characters.

```
vector1 <- c("5", "12", "7", "32")
```

The result of the command will be 7 because it's sorted alphabetically.

```
max(vector1)
```

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

The vector is sorted alphabetically (not using the numeric value of each element), so the result is: 12, 32, 5, 7.

```
sort(vector1)
```

```
## [1] "12" "32" "5"  "7"
```

The command produces an error because it's not possible to sum a vector which elements are characters.

```
sum(vector1)
```

b. Series of commands.

The commands don't work. Because the first element of the vector is a character, all the elements in the vector are changed to the most "generic" type of data of its elements which in this case is characters. So when we try to add the second and third element, R tells us it's not possible to add non-numeric arguments, because they're characters.

```
vector2 <- c("5", 7, 12)
vector2[2] + vector2[3]
```

The commands work and the result is 19 (7+12). A mainframe can contain different kind of data types, so even the first element is a character, the other elements are number and adding the second and third element works fine.

```
dataframe3 <- data.frame(z1 = "5", z2 = 7, z3 = 12)
dataframe3[1,2] + dataframe3[1,3]
```

```
## [1] 19
```

Each element of a list is a list, when we do list4[2] we get the second element of the list (with its name and value), while doing list4[[2]] returns the value of the element. This is why the second command in the code chunk works while the last one doesn't.

```
list4 <- list(z1 = "6", z2 = 42, z3 = "49", z4 = 126)
list4[[2]] + list4[[4]]
list4[2] + list4[4]
```

c. Using command `seq`.

From 1 to 10000 in increments of 372.

```
seq(from = 1,
    to = 10000,
    by = 372)
```

```
##  [1]    1  373  745 1117 1489 1861 2233 2605 2977 3349 3721 4093 4465 4837
## [15] 5209 5581 5953 6325 6697 7069 7441 7813 8185 8557 8929 9301 9673
```

Sequence of 50 numbers from 1 to 10000.

```
seq(from = 1,
    to = 10000,
    length.out = 50)
```

```
##  [1]     1.0000   205.0612   409.1224   613.1837   817.2449  1021.3061
##  [7]  1225.3673  1429.4286  1633.4898  1837.5510  2041.6122  2245.6735
## [13]  2449.7347  2653.7959  2857.8571  3061.9184  3265.9796  3470.0408
## [19]  3674.1020  3878.1633  4082.2245  4286.2857  4490.3469  4694.4082
## [25]  4898.4694  5102.5306  5306.5918  5510.6531  5714.7143  5918.7755
## [31]  6122.8367  6326.8980  6530.9592  6735.0204  6939.0816  7143.1429
## [37]  7347.2041  7551.2653  7755.3265  7959.3878  8163.4490  8367.5102
## [43]  8571.5714  8775.6327  8979.6939  9183.7551  9387.8163  9591.8776
## [49]  9795.9388 10000.0000
```

d. Using command `rep`.

This command repeats the whole vector 3 times in order.

```
rep(1:3,
    times = 3)
```

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

While this command repeats each element of the vector 3 times (before passing on to the next element).

```
rep(1:3,
    each = 3)
```

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

Part II: "Massage" your data.

a. Reading with `read.delim`

We used the following 4 parameters.

1. **file:** The name of the file you want to read.
2. **header:** Binary parameter. If TRUE then the first *row* of the file is the header. Conversely, if it's FALSE then the first row of the file isn't a header.
3. **sep:** The character separator in our file. In our case, the character tab ("\t").
4. **stringsAsFactors:** Binary parameter. If TRUE then every string column will be transformed into data type factor, otherwise it'll remain as character.

```
youraw = read.delim(file = "you.tsv",
                    header = TRUE,
                    sep = "\t",
                    stringsAsFactors = FALSE)

str(object = youraw)
```

```
## 'data.frame':    68 obs. of  6 variables:
##  $ Where.are.you.from.                                              : chr  "Spain" "Italy" "Italy" "I
##  $ Previous..academic..life.                                        : chr  "Physics" "Statistics" "Sta
##  $ How.well.do.you.know..Probability                                : int  3 4 4 2 2 4 2 2 3 3 ...
##  $ How.well.do.you.know..Statistics                                 : int  3 4 4 2 2 2 4 4 3 2 ...
##  $ How.well.do.you.know..R                                          : int  1 4 3 3 2 0 3 0 4 4 ...
##  $ Besides.R..do.you.fluently.know.use.other.programming.languages.: chr  "Yes, C++, Matlab, Bash" "
```

b. How many rows and columns does *youraw* have?

It has 68 rows and 6 columns.

```
# Number of rows
nrow(youraw)
```

```
## [1] 68
```

```r
# Number of columns.
ncol(youraw)
```

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

c. What are the names of the columns of youraw?

The names are:

1. "Where.are.you.from"
2. "Previous..academic..life"
3. "How.well.do.you.know..Probability"
4. "How.well.do.you.know..Statistics"
5. "How.well.do.you.know..R"
6. "Besides.R..do.you.fluently.know.use.other.programming.languages."

```r
names(youraw)
```

```
## [1] "Where.are.you.from."
## [2] "Previous..academic..life."
## [3] "How.well.do.you.know..Probability"
## [4] "How.well.do.you.know..Statistics"
## [5] "How.well.do.you.know..R"
## [6] "Besides.R..do.you.fluently.know.use.other.programming.languages."
```

d. What is the value of row 7, column 5 of youraw?

The value is 3.

```r
youraw[7, 5]
```

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

e. Display the second row of youraw in its entirety.

```r
youraw[2, ]
```

```
##   Where.are.you.from. Previous..academic..life.
## 2               Italy                Statistics
##   How.well.do.you.know..Probability How.well.do.you.know..Statistics
## 2                                 4                                 4
##   How.well.do.you.know..R
## 2                       4
##   Besides.R..do.you.fluently.know.use.other.programming.languages.
## 2                                                      Yes, Java, SQL
```

f. Now explain what this command does:

```
youclean <- youraw
names(youclean)
```

```
## [1] "Where.are.you.from."
## [2] "Previous..academic..life."
## [3] "How.well.do.you.know..Probability"
## [4] "How.well.do.you.know..Statistics"
## [5] "How.well.do.you.know..R"
## [6] "Besides.R..do.you.fluently.know.use.other.programming.languages."
```

We make a copy of the list and called it `youclean`. Then, we run the command `names(youclean)` which changes the name of the columns given the entry array.

```
names(youclean) <- c("where", "prev.life", "good.prob", "good.stat", "good.R", "other.lan")
names(youclean)
```

```
## [1] "where"     "prev.life" "good.prob" "good.stat" "good.R"    "other.lan"
```

g. Creation of two new variables `lan1` and `lan2`

The Task

The goal is to create two new variables inside the dataset youclean named `lan1` and `lan2` which contains the first two options listed in other.lan. In case less than two options are provided, fill in the new variables with NA as needed.

Solution

Our approach consist in creating a `base` array of programming languages and use string manipulation functions to find matches of all elements in the `base` array. Then, we sort the matches and pick the first 2 elements. Finally, if there are less than 2 matches we fill the NA's as needed.

We decided this instead of different approaches for several reasons:

1. The task is to find **programming languages**. As a result, the are 7 cases where the *technology* described **are not** programming languages. These are:

    - **Turbo pascal:** Is an IDE which has a compiler for the programming language pascal. Therefore, we added pascal as a possible language in `base`.
    - **MySQL:** Is a relational database management system (RDBMS) which uses SQL that is known to be turing complete, therefore we added sql as the programming language.
    - **Android:** Is a Mobile Operating System in which you need several languages to develop applications. Therefore, we did not add a programming language with android.
    - **Windows:** Is an Operating System that can be related to multiple programming languages. In this case, we did not add a programming language
    - **LabView:** Is a System Design Software designed for a graphical programming language called (G). As a result we added G as a possible programming language.

- **CSS** and **XML**: CSS is a style sheet language used for describing the presentation of a document written in a markup language Wikipedia/CSS and XML is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable Wikipedia/XML. By their own, CSS and XML are not programming languages they can be consideredas such if they work together with HTML (Example of interaction between HTML and CSS). We decided, however, not to include them in our programming languages array `base`.

2. There is not a clear structure of the answers given by the students. We can find indeed several patterns such as:

   - They usually responded Yes or No at the beginning of the sentence.
   - They usually separated the languages with commas.
   - Sometimes, they added text before or after the programming languages.
   - Everyone wrote `C++` which may have problems with regular expressions. Therefore, we changed to `cpp`.

3. In the future, we think that if the number of students increases the chance of getting an unusual pattern is high. Therefore, with our `base` array we just need to add a new programming language to the array if we have not seen it before.

   - **Note:** The proper solution should be finding a file with *all* the programming languages and load it everytime we run our algorithm. We found this List of Programming Languages. We could have made the list but instead we decided just to put the languages used in our scope.

4. Finally, even though our solution may be in theory slower than another approaches. We expect student's answers to be short.

```
# Array of languages
languages = c("cpp", "matlab", "bash","js", "javascript","java", "sql", "c", "sas", "python", "objective

# New columns to youclean
lan1arr = character()
lan2arr = character()

# Preprocessing: Transforming languages to proper syntax.
answers = tolower(youclean$other.lan)
answers = gsub("pl/sql", "pl-sql", answers)
answers = gsub("objective c", "objective-c", answers)
answers = gsub("c\\+\\+", "cpp", answers)
answers = gsub("mysql", "sql", answers)

# Substitution of special characters to whitespace
answers = gsub("\\,", " ", answers)
answers = gsub("\\."," ", answers)
answers = gsub("/"," ", answers)
answers = gsub("!"," ", answers)

for(answer in answers){
  # Assuming minimum positions 99999
  min_1 = 99999
  min_2 = 99999

  # Default languages as NA
  lan1 = NA
  lan2 = NA
```

```
  # Split every answer in whitepaces
  for(chunk in strsplit(answer, " ")[[1]]){

    # Trim eliminates whitespaces on the left and right of string.
    chunk = trimws(chunk)

    # For every language, we search if exists in every chunk
    for(language in languages){

      # Returns the position of the occurrence of the language in the chunk, -1 otherwise
      position = regexpr(language, chunk, fixed=TRUE)

      # If the language exists in the chunk and has the same length as the chunk
       if(position[1]>-1 && nchar(chunk) == nchar(language)){

        # In case the position is lower than our minimum, update min_1 and min_2
        if(position[1] < min_1){
          min_2 = min_1
          lan2 = lan1

          min_1 = position[1]
          lan1 = language

          # If we find 1 language it's impossible to find another in the chunk.
          break
        }else if(position[1] < min_2){
          min_2 = position[1]
          lan2 = language

          # BIS
          break
        }
      }
    }
  }

  # Update arrays.
  lan1arr = c(lan1arr, lan1)
  lan2arr = c(lan2arr, lan2)
}

# Create new columns
youclean$lan1 = lan1arr
youclean$lan2 = lan2arr
```

h. Very (not) last step: Data Types.

1. Run the following code on your data and explain what it does:

   - **class:** Used to define or identify what "type" an object is from the point of view of object-oriented programming in R.
   - **typeof:** Determines the (R internal) type or storage mode of any object.

- **mode:** Get or set the type or storage mode of an object.

According to the Manual of R the main difference between **typeof** and **mode** is that *mode gives information about the mode of an object in the sense of Becker, Chambers & Wilks (1988), and is more compatible with other implementations of the S language*. Conversely, **typeof** gives the type of the object from R internal point of view.

2. Explain the difference you see between the first three and the last three lines of code.

   In the first three lines, the type, class and mode of the column **where** were the same (character). This means that every entry of every row of the column **where** is stored as plain text. In contrast, when we transformed the column with function `as.factor` we changed the R class to **factor**, the R internal type of storage to **integer** and the type of the object in the sense of S to **numeric**.

   This said, the `as.factor` function transform the object to a list of interger where every integer is mapped to a list of trings containing the original data.

```
# Before
class(youclean$where)
```

```
## [1] "character"
```

```
typeof(youclean$where)
```

```
## [1] "character"
```

```
mode(youclean$where)
```

```
## [1] "character"
```

```
# Change to factor
youclean$where <- as.factor(youclean$where)
levels(youclean$where)
```

```
##  [1] "Argentina"  "Azerbaijan" "Belgium"    "Brazil"     "China"
##  [6] "Egypt"      "Georgia"    "Germany"    "Iran"       "Italy"
## [11] "Italy "     "Montenegro" "Pakistan"   "Russia"     "Spain"
## [16] "Togo"       "Turkey"     "Turkey "    "Venezuela"
```

```
nlevels(youclean$where)
```

```
## [1] 19
```

```
# After
class(youclean$where)
```

```
## [1] "factor"
```

```
typeof(youclean$where)
```

```
## [1] "integer"
```

```
mode(youclean$where)
```

```
## [1] "numeric"
```

Recycle the relevant part of this code to turn into factors also prev.life

```
youclean$prev.life <- as.factor(youclean$prev.life)
```

i: Saving file

Using `save` command to save our data youclean and youraw.

```
save(youclean, youraw, file="you.RData")
```

Part III: Exercises from the blue book

Definitions and properties:

Some definitions and properties will be used on the exercises. Each reference will be used given the number here.

(1) $\forall E, F \in \Omega$ we have $E^c \cap F^c = (E \cup F)^c$
(2) $\forall E, F \in \Omega$ we have $E^c \cup F^c = (E \cap F)^c$
(3) $\forall A \in \Omega$ we have $\mathbb{P}(A^c) = 1 - \mathbb{P}(A)$
(4) $\forall A, B \in \Omega$ we have $\mathbb{P}(A \cup B) = \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A \cap B)$
(5) $\forall A, B \in \Omega$ with $\mathbb{P}(B) > 0$ we have $\mathbb{P}(A|B) = \dfrac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}$

1. Probability basics Chapters 2 & 3

Problem 2.2

Let E and F be two events for which one knows that the probability that at least one of them occurs is $\frac{3}{4}$. What is the probability that neither E nor F occurs? Hint: use one of DeMorgan's laws: $E^c \cap F^c = (E \cup F)^c$.

Solution:

$$\text{We know: } \mathbb{P}(E \cup F) = \frac{3}{4}. \text{ So,}$$

$$\begin{aligned} \mathbb{P}(E^c \cap F^c) =& \mathbb{P}((E \cup F)^c) \quad (1) \\ =& 1 - \mathbb{P}(E \cup F) \quad (3) \\ =& 1 - \frac{3}{4} = \frac{1}{4} \end{aligned}$$

Problem 2.6

When $\mathbb{P}(A) = \frac{1}{3}, \mathbb{P}(B) = \frac{1}{2}$ and $\mathbb{P}(A \cup B) = \frac{3}{4}$ what is:

a. $\mathbb{P}(A \cap B)$?
b. $\mathbb{P}(A^c \cup B^c)$?

Solution:

a.

$$\begin{aligned}\mathbb{P}(A \cap B) =& \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A \cup B) \quad (4)\\ =& \frac{1}{3} + \frac{1}{2} - \frac{3}{4} = \frac{1}{12} \approx 0.083\end{aligned}$$

b.

$$\begin{aligned}\mathbb{P}(A^c \cup B^c) =& \mathbb{P}((A \cap B)^c) \quad (2)\\ =& 1 - \mathbb{P}(A \cap B) \quad (3)\\ =& 1 - \frac{1}{12} = \frac{11}{12} \approx 0.917\end{aligned}$$

Problem 2.9

We toss a coin three times. For this experiment we choose the sample space:

$$\Omega = \{HHH, THH, HTH, HHT, TTH, THT, HTT, TTT\}$$

where T stands for tails and H for heads.

a. Write down the set of outcomes corresponding to each of the following events:

A : "we throw the tails exactly two times"
B : "We throw tails at least two times"
C : "tails did not appear before a head appeared"
D : "The first result throw results in tails"

b. Write down the set of outcomes corresponding to each of the following events: A^c, $A \cup (C \cap D)$, and $A \cap D^c$.

Solution

a.

$A = \{HTT, THT, TTH\}$
$B = \{HTT, THT, TTH, TTT\}$
$C = \{HHH, HTT, HTH\}$
$D = \{THH, THT, TTT, TTH\}$

b.

$$\begin{aligned}A^c =& \Omega - A\\ =& \{HHH, THH, HTH, HHT, TTH, THT, HTT, TTT\} - \{HTT, THT, TTH\}\\ =& \{HHH, THH, HTH, HHT, TTT\}\\ A \cup (C \cap D) =& \{HTT, THT, TTH\} \cup (\{HHH, HTT, HTH\} \cap \{THH, THT, TTT, TTH\})\\ =& \{HTT, THT, TTH\} \cup \emptyset\\ =& \{HTT, THT, TTH\}\\ A \cap D^c =& \{HTT, THT, TTH\} \cap \{HHH, HTT, HTH, HHT\}\\ =& \{HTT\}\end{aligned}$$

Problem 3.5

A ball is drawn at random from an urn containing one red and one white ball. If the white ball is drawn, it is put back into the urn. If the red ball is drawn, it is returned to the urn together with two more red balls. Then a second draw is made. What is the probability a red ball was drawn on both the first and the second draws?

Solution Given the experiment, we have the events:

$R_1 = \{\text{Draw of red ball in first attempt}\}$
$R_2 = \{\text{Draw of red ball in second attempt}\}$

We know that: $\mathbb{P}(R_1) = \frac{1}{2}$ and $\mathbb{P}(R_2|R_1) = \frac{3}{4}$

We need to calculate: $\mathbb{P}(R_1 \cap R_2)$

$$\begin{aligned}\mathbb{P}(R_1 \cap R_2) =& \mathbb{P}(R_1) \cdot \mathbb{P}(R_2 \mid R_1) \qquad (5)\\ &= \frac{1}{2} \cdot \frac{3}{4} = \frac{3}{8} = 0.375\end{aligned}$$

Problem 3.10

A student takes a multiple-choice exam. Suppose for each question he either knows the answer or gambles and chooses an option at random. Further suppose that if he knows the answer, the probability of a correct answer is 1, and if he gambles this probability is $\frac{1}{4}$. To pass, students need to answer at least 60% of the questions correctly. The student has studied for a minimal pass, i.e., with probability 0.6 he knows the answer to a question. Given that he answers a question correctly, what is the probability that he actually knows the answer?

Solution We have the events:

C : “Answering a question correctly"
K : “Knowing the answer to a question""

We know that: $\mathbb{P}(K) = 0.6 \rightarrow \mathbb{P}(K^c) = 0.4$, $\mathbb{P}(C \mid K) = 1$ and $\mathbb{P}(C \mid K^c) = \frac{1}{4}$

We need to calculate: $\mathbb{P}(K \mid C)$

$$\begin{aligned}\mathbb{P}(K \mid C) =& \frac{\mathbb{P}(K) \cdot \mathbb{P}(C \mid K)}{\mathbb{P}(C)}\\ =& \frac{\mathbb{P}(K) \cdot \mathbb{P}(C \mid K)}{\mathbb{P}(K) \cdot \mathbb{P}(C \mid K) + \mathbb{P}(K^c) \cdot \mathbb{P}(C \mid K^c)}\\ =& \frac{1 \cdot 0.6}{1 \cdot 0.6 + 0.25 \cdot 0.4} = \frac{6}{7} \approx 0.857\end{aligned}$$

Problem 3.11

A breath analyzer, used by the police to test whether drivers exceed the legal limit set for the blood alcohol percentage while driving, is known to satisfy:

$$\mathbb{P}(A \mid B) = \mathbb{P}(A^c \mid B^c) = p$$

where A is the event breath analyzer indicates that legal limit is exceeded and B drivers blood alcohol percentage exceeds legal limit. On Saturday night about 5% of the drivers are known to exceed the limit.

a. Describe in words the meaning of $\mathbb{P}(B^c \mid A)$.
b. Determine $\mathbb{P}(B^c \mid A)$ if $p = 0.95$.
c. How big should p be so that $\mathbb{P}(A \mid B) = 0.9$.

Solution

a. $\mathbb{P}(B^c \mid A)$ = "Probability that the blood of the driver doesn't exceed the legal limit given that the breath analyzer indicates it does"

b. We have the events:

A : "breath analyzer indicates that legal limit is exceeded"
B : "drivers blood alcohol percentage exceeds legal limit."

We know that: $\mathbb{P}(A \mid B) = \mathbb{P}(A^c \mid B^c) = p$ Therefore, $\mathbb{P}(A^c \mid B) = \mathbb{P}(A \mid B^c) = 1 - p$ using (3)

With $p = 0.05$ we have $\mathbb{P}(A \mid B) = \mathbb{P}(A^c \mid B^c) = 0.05$ and $\mathbb{P}(A^c \mid B) = \mathbb{P}(A \mid B^c) = 0.95$

Assuming that behaviour on Saturdays remains in the whole week we have that: $\mathbb{P}(B) = 0.05$ and $\mathbb{P}(B^c) = 0.95$

We need to calculate: $\mathbb{P}(B^c \mid A)$

$$\begin{aligned}
\mathbb{P}(B^c \mid A) =& \frac{\mathbb{P}(A \mid B^c) \cdot \mathbb{P}(B^c)}{\mathbb{P}(A)} \\
=& \frac{\mathbb{P}(A \mid B^c) \cdot \mathbb{P}(B^c)}{\mathbb{P}(A \mid B) \cdot \mathbb{P}(B) + \mathbb{P}(A \mid B^c) \cdot \mathbb{P}(B^c)} \\
=& \frac{(1 - \mathbb{P}(A^c \mid B^c)) \cdot (1 - \mathbb{P}(B))}{\mathbb{P}(A \mid B) \cdot \mathbb{P}(B) + (1 - \mathbb{P}(A^c \mid B^c)) \cdot (1 - \mathbb{P}(B))} \\
=& \frac{(1 - 0.95) \cdot (1 - 0.05)}{0.95 \cdot 0.05 + (1 - 0.95) \cdot (1 - 0.05)} = 0.5
\end{aligned}$$

c. Assuming same scenario than in b. (but $p = 0.95$) we now need to find how big should p be so that $\mathbb{P}(B \mid A) = 0.9$.

We need p such that $\mathbb{P}(B \mid A) = 0.9$

$$
\begin{aligned}
\mathbb{P}(B \mid A) &= \frac{\mathbb{P}(A \mid B) \cdot \mathbb{P}(B)}{\mathbb{P}(A)} \\
0.90 &= \frac{\mathbb{P}(A \mid B) \cdot \mathbb{P}(B)}{\mathbb{P}(A \mid B) \cdot \mathbb{P}(B) + \mathbb{P}(A \mid B^c) \cdot \mathbb{P}(B^c)} \\
0.90 &= \frac{p \cdot \mathbb{P}(B)}{p \cdot \mathbb{P}(B) + (1-p) \cdot \mathbb{P}(B^c)} \\
0.90 &= \frac{p \cdot 0.05}{p \cdot 0.05 + (1-p) \cdot 0.95} \\
p \cdot 0.05 &= 0.90 \cdot (p \cdot 0.05 + (1-p) \cdot 0.95) && \text{because } p \cdot 0.05 + (1-p) \cdot 0.95 > 0 \\
0.05 \cdot p &= 0.90 \cdot (-0.90 \cdot p + 0.95) \\
0.05 \cdot p &= -0.81 \cdot p + 0.855 \\
0.05 \cdot p + 0.81 \cdot p &= 0.855 \\
0.86 \cdot p &= 0.855 \\
p &= \frac{0.855}{0.86} = \frac{855}{860} = \frac{171}{172}
\end{aligned}
$$

d. Therefore, p should be $\frac{171}{172}$. Let's check:

$$
\mathbb{P}(B \mid A) = \frac{\frac{5 \cdot 171}{100 \cdot 172}}{\frac{5 \cdot 171}{100 \cdot 172} + \frac{95 \cdot 1}{100 \cdot 172}} = \frac{855}{950} = \frac{9}{10}
$$

Problem 3.18

Suppose A and B are events with $0 < \mathbb{P}(A) < 1$ and $0 < \mathbb{P}(B) < 1$.

a. If A and B are disjoint, can they be independent?
b. If A and B are independent, can they be disjoint?
c. If $A \subset B$, can A and B be independent?
d. If A and B are independent, can A and $A \cup B$ be independent?

Solution

a. Let A and B disjoint events such that $0 < \mathbb{P}(A) < 1$ and $0 < \mathbb{P}(B) < 1$. So, let's assume that A and B are independent. $\mathbb{P}(A \cap B) = \mathbb{P}(A) \cdot \mathbb{P}(B)$. Therefore, to be independent it must satisfy $\mathbb{P}(A \cap B) = \mathbb{P}(\emptyset) = 0$ because $A \cap B = \emptyset$. Then, $\mathbb{P}(A) \cdot \mathbb{P}(B) = 0$. But we know that $\mathbb{P}(A) > 0$, $\mathbb{P}(B) > 0$ then $\mathbb{P}(A) \cdot \mathbb{P}(B) > 0$. As a result, if A and B are disjoint, $0 < \mathbb{P}(A) < 1$ and $0 < \mathbb{P}(B) < 1$ they can't be independent.

b. Let A and B independent events such that $0 < \mathbb{P}(A) < 1$ and $0 < \mathbb{P}(B) < 1$. So, if A and B are independent then $\mathbb{P}(A \cap B) = \mathbb{P}(A) \cdot \mathbb{P}(B)$. We know that $\mathbb{P}(A) > 0$ and $\mathbb{P}(B) > 0$ then $\mathbb{P}(A) \cdot \mathbb{P}(B) > 0$. Which means, $A \cap B \neq \emptyset$. Finally, if A and B are independent, $0 < \mathbb{P}(A) < 1$ and $0 < \mathbb{P}(B) < 1$ they can't be disjoint.

c. Let A and B events such that $A \subset B$ and $0 < \mathbb{P}(A) < 1$ and $0 < \mathbb{P}(B) < 1$. For A and B to be independent it must be satisfied that $\mathbb{P}(A \cap B) = \mathbb{P}(A) \cdot \mathbb{P}(B)$ and we know that if $A \subset B$ then $A \cap B = A$. Therefore using both premises we have $\mathbb{P}(A) \cdot \mathbb{P}(B) = \mathbb{P}(A)$ which means, as $\mathbb{P}(A) > 0$, that $\mathbb{P}(B) = 1$. But, by definition $\mathbb{P}(B) < 1$. Finally, if $A \subset B$, $0 < \mathbb{P}(A) < 1$ and $0 < \mathbb{P}(B) < 1$ they can't be independent.

d. Let A and B independent events such that $0 < \mathbb{P}(A) < 1$ and $0 < \mathbb{P}(B) < 1$. So, $\mathbb{P}(A \cap (A \cup B)) = \mathbb{P}((A \cap A) \cup (A \cap B)) = \mathbb{P}(A) + \mathbb{P}(A \cap B) - \mathbb{P}(A \cap B) = \mathbb{P}(A)$. Let's assume that A and $A \cup B$ are independent then $\mathbb{P}(A \cap (A \cup B)) = \mathbb{P}(A) \cdot \mathbb{P}(A \cup B)$. We now have that $\mathbb{P}(A) \cdot \mathbb{P}(A \cup B) = \mathbb{P}(A)$ which means $\mathbb{P}(A \cup B) = 1$ and $A \cup B = \Omega$ because $\mathbb{P}(A) > 0$. Now, let's use the probability of a union:

$$\begin{aligned}
\mathbb{P}(A \cup B) =& \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A \cap B) & \\
1 =& \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A \cap B) & A \cup B = \Omega \\
1 =& \mathbb{P}(A) + \mathbb{P}(B) - \mathbb{P}(A) \cdot \mathbb{P}(B) & A \text{ and } B \text{ independent} \\
1 - \mathbb{P}(A) =& \mathbb{P}(B) - \mathbb{P}(A) \cdot \mathbb{P}(B) & \\
1 - \mathbb{P}(A) =& \mathbb{P}(B) \cdot (1 - \mathbb{P}(A)) & \text{Distributive} \\
\mathbb{P}(A^c) =& \mathbb{P}(B) \cdot \mathbb{P}(A^c) & (3) \\
1 =& \mathbb{P}(B) & \mathbb{P}(A^c) > 0
\end{aligned}$$

But, by definition $\mathbb{P}(B) < 1$. Therefore, if A and B are independent, $0 < \mathbb{P}(A) < 1$ and $0 < \mathbb{P}(B) < 1$ then A and $A \cup B$ can't be independent.

2. Topic: Discrete random variables Chapter 4

Problem 4.2

Let X be a discrete random variable with probability mass function p given by:

a	-1	0	1	2
$p(a)$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{2}$

and $p(a) = 0$ for all other a.

a. Let the random variable Y be dened by $Y = X^2$, i.e., if $X = 2$,then $Y = 4$. Calculate the probability mass function of Y.
b. Calculate the value of the distribution functions of X and Y in $a = 1$, $a = \frac{3}{4}$ and $a = \pi - 1$.

Solution

a. As we define $Y = X^2$, we evaluate every element of the support of X to obtain the image in Y.

- if $X = -1$ then $Y = 1$.
- if $X = 0$ then $Y = 0$.
- if $X = 1$ then $Y = 1$
- if $X = 2$ then $Y = 4$

Therefore, we name the probability mass function of Y as $p_Y(a), a = \{0, 1, 4\}$

- $p_Y(0) = p(0) = \frac{1}{8}$
- $p_Y(1) = p(-1) + p(1) = \frac{1}{4} + \frac{1}{8} = \frac{3}{8}$
- $p_Y(4) = p(2) = \frac{1}{2}$

a	0	1	4
$p_Y(a)$	$\frac{1}{8}$	$\frac{3}{8}$	$\frac{1}{2}$

b. First, we calculate de distribution function of X and Y. We named each as $F_X(a)$ and $F_Y(a)$ for X and Y respectively:

a	-1	0	1	2
$F_X(a) = \mathbb{P}(X \leq a)$	$\frac{1}{4}$	$\frac{3}{8}$	$\frac{1}{2}$	1

a	0	1	4
$F_Y(a) = \mathbb{P}(Y \leq a)$	$\frac{1}{8}$	$\frac{1}{2}$	1

- $F_X(X \leq 1) = \frac{1}{2}$
- $F_Y(Y \leq 1) = \frac{1}{2}$
- $F_X\left(X \leq \frac{3}{4}\right) = \frac{3}{8}$
- $F_Y\left(Y \leq \frac{3}{4}\right) = \frac{1}{8}$
- $F_X(X \leq \pi - 3) = F_X(X \leq 0.1415) = \frac{3}{8}$
- $F_Y(Y \leq \pi - 3) = F_X(Y \leq 0.1415) = \frac{1}{8}$

Problem 4.11

You decide to play monthly in two different lotteries, and you stop playing as soon as you win a prize in one (or both) lotteries of at least one million euros. Suppose that every time you participate in these lotteries, the probability to win one million (or more) euros is p_1 for one of the lotteries and p_2 for the other. Let M be the number of times you participate in these lotteries until winning at least one prize. What kind of distribution does M have, and what is its parameter?

Solution We have defined M as the number of times you participate in these lotteries until winning at least one prize. We assume the probability of winning one contest is independent to the probability of winning the other and the probability of winning in any month is independent of the previous ones.

Given the conditions, M has a **geometric** distribution (Number of trials until the first success).

The parameter of the **geometric** distribution is the probability of success p.

Lets define events. W_1 : " Win at least 1.000.000 euros in lottery 1" and W_2 : " Win at least 1.000.000 euros in lottery 2" and assign probabilities p_1 and p_2 such that: $\mathbb{P}(W_1) = p_1$ and $\mathbb{P}(W_2) = p_2$.

The probability of failure is $\mathbb{P}(W_1^c \cap W_2^c) = \mathbb{P}(W_1^c) \cdot \mathbb{P}(W_2^c) = (1 - p_1) \cdot (1 - p_2)$

The parameter (probability of success) p can be expressed as:

1. $p = 1 - \mathbb{P}(\text{failure}) = 1 - (1 - p_1) \cdot (1 - p_2)$
2. $p = \mathbb{P}(W_1 \cup W_2) = \mathbb{P}(W_1) + \mathbb{P}(W_2) - \mathbb{P}(W_1 \cap W_2) = p_1 + p_2 - p_1 \cdot p_2$
3. $p = \mathbb{P}(W_1 \cap W_2) + \mathbb{P}(W_1 \cap W_2^c) + \mathbb{P}(W_1^c \cap W_2) = p_1 \cdot p_2 + p_1(1 - p_2) + p_2(1 - p_1)$

Finally, as an example the probability mass distribution associated to M ~Geom$(1 - (1 - p_1) \cdot (1 - p_2))$ using the first parameter is:

$$p_M(m) = ((1 - p_1) \cdot (1 - p_2))^{m-1} \cdot (1 - (1 - p_1) \cdot (1 - p_2)) \text{ for } m = 1, 2, \cdots$$

Problem 4.14

We throw a coin until a head turns up for the second time, where p is the probability that a throw results in a head and we assume that the outcome of each throw is independent of the previous outcomes. Let X be the number of times we have thrown the coin.

Solution

a. We can get $\mathbb{P}(X = 2)$ only if we get one head in the first toss and another head in the second toss. Since the probability of getting a head is p, then: $\mathbb{P}(X = 2) = p * p = p^2$ In addition, to have 2 heads in 3 tosses, it is necessary to get a tail before the second head. We have two possibilities $\{HTH, THH\}$. So, $\mathbb{P}(X = 3) = p(1-p)p + (1-p)p^2 = 2(1-p)p^2$. Finally, to get 2 heads in 4 tosses, we need to have 2 tails before the second head: $\{HTTH, TTHH, THTH\}$. Which gives us:$\mathbb{P}(X = 4) = p(1-p)(1-p)p + (1-p)(1-p)p^2 + (1-p)p(1-p)p = 3(1-p)^2p^2$

b. We know that we throw the coin until the second head appears. Given n the number of throws, we will always have 2 heads and $n-2$ tails. The probability of each event that has 2 heads and $n-2$ tails is given by $p^2 \cdot (1-p)^{n-2}$ assuming independence in each throw. Lastly, we need to count how many of these events exist. To do this, is important to notice that we end our experiment when we get our second head, as a result, the last throw will **always** be a head. Therefore, we need to count the ways 1 head and $n-2$ tails can be obtained in the first $n-1$ throws. This problem is equivalent to assign a position to the head given the $n-1$ spots available. This said, we need the number of subsets of size 1 from a set of size $n-1$ which is $\binom{n-1}{1} = n-1$. Finally, we have that $P(X = n) = (n-1)p^2(1-p)^{n-2}$ with $n \geq 2$.

Part IV: Coins, Randomness and Genetics

Summary of the video

The video starts talking about how many times do we need to shuffle the deck of cards to make it random, which means, the cards before the shuffle are totally different just after. As we know, the smaller the deck, the easier is tho get it shuffled randomly, because if the deck is 4 cards, them the number of possibilities is 4! = 4x3x2 = 24.

But usually we cannot try out all the possibilities. In a complete deck of 52 cards, it means that we have 52! possibilities of sequences of cards, which results in more than $8x10^{67}$ possibilities.

Also, they mentioned that if you give an ordered deck of cards to a friend and tell him to shuffle by the common way of shuffling (without seeing while he does), which divides the deck in two parts of almost the same size, and shuffle them and put them together again, you will still have $2^{number\_of\_shuffles}$ rising 'small' sequences in the middle of the deck. If your friend mix 3 times, so there's 8 rising sequences in the deck. And then you do a magic: remove cards one by one. If a card is not in order with the previous card, you start another pile. If it's in order, you put in the same pile. In the 9th pile is the card that the person has taken.

He affirmed again, practically, that the deck is never completely mixed, it is not random. He raised a question: how much do we accept it to be shuffled? It will not be random, but how much do we consider it enough?

When he cites the on-line poker, he compares that to have a deck really shuffled randomly, you would need to have computer with the capacity to process numbers in the order of $6x10^{67}$, which is far from the possibilities of a computer of 32 bits ($2^{32} = 4,3$ billions possibilities).

Then, a student asked how about to put the cards on the table and mash them with the hands. He explained that this is efficient and is hard to predict the result or the efficiency of this method because it is hard to model this method, but certainly, with 8x10^67 possibilities, there will be still cards in sequence.

Another student suggested to take off the top ten cards, and shuffle and then shuffle the rest of the deck. The professor explained that there are many machines in Vegas which use this idea: the deck enters in the machine and cards are separated in ten shelves, some on and others under the pile. But again, the problem resides that the decision that the machine does is based in the random number which it generates, again 2^{32} or 2^{64}.

He explained again that practically nothing is random because we do not try all the possibilities: we do not shuffle the cards enough, we do not toss the coin enough, etc.

He stated that the problem of lack of randomness in tossing coins is because we use Physics, we can define the position in which the coin will end. If we know the initial condition (head or tail), apply a known force in a known region of the coin, you can know the height that the coin will reach and how many turns it will spin, so you can tell how it will end - tail or head.

So, he concludes that nothing is completely random.

Coin tosses

Now imagine tossing a coin successively, and waiting till the first time a particular pattern appears, say [HTT]. For example, if the sequence of tosses was:

HHTHHTH[HTT]HHTTTHTH

The pattern [HTT] would first appear after the 10th toss.

Now, consider these two different patterns [HTH] and [HTT].

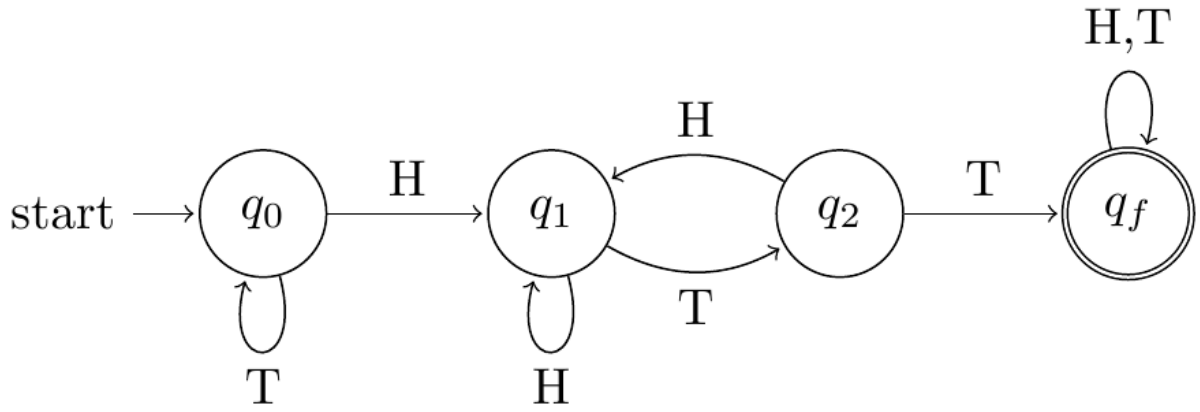
In your opinion which of the following three options is true:

1. The average number of tosses until [HTH] is **larger** than that until [HTT].
2. The average number of tosses until [HTH] is **the same** as that until [HTT].
3. The average number of tosses until [HTH] is **smaller** than that until [HTT].

Solution

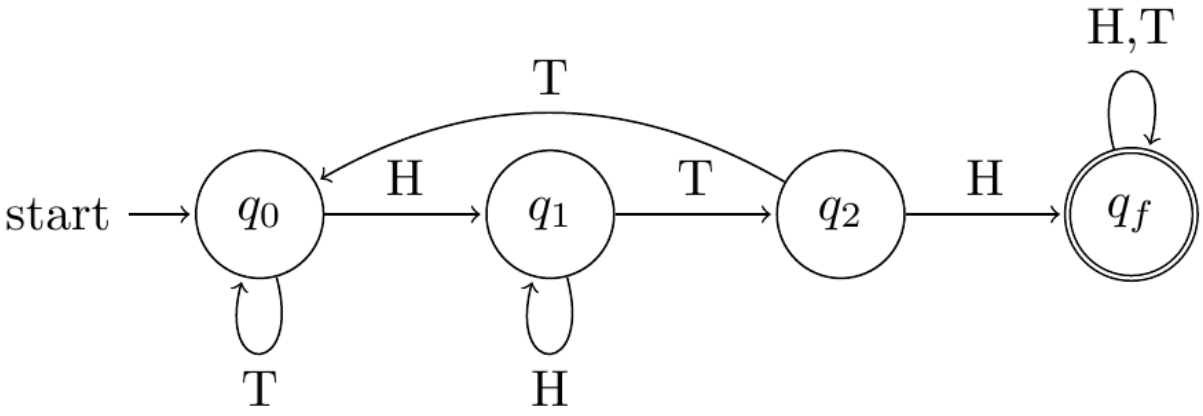
At first sight, it seems that the answer should be **the same** because, assuming independence and a balanced coin, the probability of both patterns is the same: $\frac{1}{8}$. However, we'll show in 3 different ways that the average number of tosses until [HTH] is actually **larger** than that until [HTT].

A computational approach We can view the problem of finding the patterns as the Substring Search problem. There exist several implementations to solve these problem. One of them, consists in obtaining a Deterministic finite state automaton (DFA) which accepts the string if the sequence of transitions reading every character leads to halt state (or final state). We now show both automaton in our context:



This is the automaton for string [HTT]. Assuming the alphabet {H,T}. Every state transitions to other state if we read H or T.

1. In the initial state, q_0 we assume the empty string.
2. In q_1 we assume that the last character we have read is H
3. In q_2 we assume that the last two characters we have read are HT
4. Finally, in q_f we assume that we have read our 3 consecutive characters HTT which means we arrived to our halt state.



This is the automaton for string [HTH]. Assuming the alphabet {H,T}. Every state transitions to other state if we read H or T.

1. In the initial state, q_0 we assume the empty string.
2. In q_1 we assume that the last character we have read is H
3. In q_2 we assume that the last two characters we have read are HT
4. Finally, in q_f we assume that we have read our 3 consecutive characters HTT which means we arrived to our halt state.

It is important to notice that the difference between both automatons is on state q_2. In the case of [HTT], if we failed to get the second consecutive T we return to state q_1. Conversely, if we failed to obtain the H we return to state q_0.

First conclussion:

Analyzing these automatons, it is possible to conclude that the average number of tosses for string [HTH] is larger than for string [HTT] because when both fail in state q_2, [HTH] returns to the initial state q_0 and [HTT] only goes back to q_1.

"Simulating" our automatons: Our first conclussion is only an naive guess. To support it with empirical evidence, we decided to program both automatons and analyze the average number of tosses until we get the desired string. They're programmed in Python and should run in Rmarkdown but we've decided to put the parameter `eval=FALSE`.

Normally, this approach is called Monte Carlo Simulation. Basically, we're repeating the same experiment N (N is 20000 in the code) times. We store the number of tosses until we get our pattern in an array and then we retrieve the average number and the variance of tosses.

```
from random import random
from numpy import mean, std, var, float64

# Automata for 'HTT'
# This should be an algorithm that creates the automata given string
automata = dict()
automata[('q0','-')] = 'q0'
automata[('q0','H')] = 'q1'
automata[('q0','T')] = 'q0'
automata[('q1','H')] = 'q1'
automata[('q1','T')] = 'q2'
automata[('q2','H')] = 'q1'
automata[('q2','T')] = 'qf'

# Experiment #1
experiments = 20000
HTT = []

for exp in range(experiments):
    state = ('q0','-')
    steps = 0
    while automata[state] != 'qf':
        steps += 1
        coin = 'H' if random() < 0.5 else 'T'
        state = (automata[state], coin)
    HTT.append(steps)

# Shame on Donald Knuth
print("HTT: ",mean(HTT), var(HTT))
```

```
from random import random
from numpy import mean, std, var, float64
# Automata for 'HTH'
# This should be an algorithm that creates the automata given string

automata = dict()
automata[('q0','-')] = 'q0'
automata[('q0','H')] = 'q1'
automata[('q0','T')] = 'q0'
automata[('q1','H')] = 'q1'
automata[('q1','T')] = 'q2'
```

```
automata[('q2','H')] = 'qf'
automata[('q2','T')] = 'q0'

# Experiment #1
experiments = 200000
HTH = []

for exp in range(experiments):
    state = ('q0','-')
    steps = 0
    while automata[state] != 'qf':
        steps += 1
        coin = 'H' if random() < 0.5 else 'T'
        state = (automata[state], coin)
    HTH.append(steps)

# Shame on Donald Knuth
print('HTH: ',mean(HTH, dtype=float64), var(HTH, dtype=float64))
```

One run of both algorithms may show an output like

1. ('HTT:', 8.0010999999999992, 24.029398789999998)
2. ('HTH:', 10.01468, 58.009584497599995)

Second conclussion:

If we run our codes multiple times we concluded that the average number of tosses until we get the string [HTT] is ≈ 8 and the variance is ≈ 24. However, for string [HTH] the average number of tosses is ≈ 10 and the variance is ≈ 58. We then have a strong empirical evidence that the average number of tosses until string [HTH] is larger than until [HTT].

A markov chain approach. We may analyze our automaton as a Markov process in which the probability of transition between every state is the probability of obtaining characters H or T. The transition matrices for both automatons are:

$$\mathbf{P_{HTH}} = \begin{array}{c} \\ q_0 \\ q_1 \\ q_2 \\ q_f \end{array} \begin{array}{c} \begin{array}{cccc} q_0 & q_1 & q_2 & q_f \end{array} \\ \left\| \begin{array}{cccc} \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{array} \right\| \end{array}$$

$$\mathbf{P_{HTT}} = \begin{array}{c|cccc} & q_0 & q_1 & q_2 & q_f \\ \hline q_0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ q_1 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \\ q_2 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ q_f & 0 & 0 & 0 & 1 \end{array}$$

In both cases, we have an absorbent state q_f in which the probability of staying in the state is 1 when we arrive to this state and transient states $\{q_0, q_1, q_2\}$. Read Classification of states for further information about classification of states.

This said, we investigated and this type of matrices are called Absorbent Markov Chains. In the chapter III of Finite Markov Chains,(Kennedy and Snell, 1976, Springer-Velag). We found several theorems which lead to 3 important calculations:

1. The fundamental matrix N is obtained using $N = (I_t - Q)^{-}1$. Where Q is the transition matrix of the transient states $\{q_0, q_1, q_2\}$.
2. The expected number of steps before being absorbed when starting in transient state i is the ith entry of the vector: $t = N \cdot 1$ with 1 a 1 column vector.
3. The variance on the number of steps before being absorbed when starting in transient state i is the ith entry of the vector: $2\,(2N - I_t)\,t - t_{sq}$ where t_{sq} is the Hadamard Product of t with itself.

Now, lets see the calculations for the string [HTT]:

$$Q_{HTT} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 \end{pmatrix}$$

1. $N_{HTT} = (I_t - Q_{HTT})^{-1} = \begin{pmatrix} 2 & 4 & 2 \\ 0 & 4 & 2 \\ 0 & 2 & 2 \end{pmatrix}$

2. $t_{HTT} = N_{HTT} \cdot 1 = \begin{pmatrix} 2 & 4 & 2 \\ 0 & 4 & 2 \\ 0 & 2 & 2 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 8 \\ 6 \\ 4 \end{pmatrix}$

3. $$\begin{aligned} V_{HTT} &= (2N - I_t)\, t - t_{sq} \\ &= \left(2 \begin{pmatrix} 2 & 4 & 2 \\ 0 & 4 & 2 \\ 0 & 2 & 2 \end{pmatrix} - \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\right) \cdot \begin{pmatrix} 8 \\ 6 \\ 4 \end{pmatrix} - \begin{pmatrix} 64 \\ 36 \\ 16 \end{pmatrix} \\ &= \begin{pmatrix} 24 \\ 22 \\ 20 \end{pmatrix} \end{aligned}$$

As we can see, the average number of tosses for string [HTT] when we start in state q_0 is 8. Also, the variance is 24. If we do similar calculations for the markov chain related to string [HTH], we get that the average number of tosses is 10 and the variance is 58.

Third and final conclussion:

We got similar numbers between our simulation and the actual theoretical number using Absorbent Markov Chains. This said, we arrived to the same conclussion in 3 different ways which is, surprisingly, that **the average number of tosses until [HTH] is larger than until [HTT] (It's actually 10 vs 8).

Part V: A naive version of the Naive Bayes Classifier

The exercise gives us a table of the historical events of four weather parameters (outlook, temperature, humidity, wind) and, based in these weather conditions, we played tennis or not. Then, knowing the weather condition for today, it asks us if we will play tennis. In other words, based in our past experience with the given condition, i.e. Outlook = Sunny, Temperature = Cool, Humidity = High , Wind = Strong, what is most propable evidence expected?

We start to compute the total "Yes" and "No" probability. In overall table, how many times we played tennis and how many we did not:

```
P(Play = Yes) = 9/14
P(Play = No) = 5/14
```

Considering this results, we calculate how many times we played tennis and how many we didn't for each hypothesis. For example, the conditional probability of playing tennis in a sunny day is:

```
P(Outlook = Sunny | Play = Yes) = 2/9
P(Outlook = Sunny | Play = No) = 3/5
```

For the others hypothesis, we have:

Outlook

```
P(Outlook = Overcast | Play = Yes) = 4/9
P(Outlook = Overcast | Play = No) = 0/5
P(Outlook = Rain | Play = Yes) = 3/9
P(Outlook = Rain | Play = No) = 2/5
```

Temperature

```
P(Temperature = Hot | Play = Yes) = 2/9
P(Temperature = Hot | Play = No) = 2/5
P(Temperature = Mild | Play = Yes) = 4/9
P(Temperature = Mild | Play = No) = 2/5
P(Temperature = Cool | Play = Yes) = 3/9
P(Temperature = Cool | Play = No) = 1/5
```

Humidity

```
P(Humidity = High | Play = Yes) = 3/9
P(Humidity = High | Play = No) = 4/5
P(Humidity = Normal | Play = Yes) = 6/9
P(Humidity = Normal | Play = No) = 1/5
```

Wind

```
P(Wind = Strong | Play = Yes) = 3/9
P(Wind = Strong | Play = No) = 3/5
P(Wind = Weak | Play = Yes) = 6/9
P(Wind = Weak | Play = No) = 2/5
```

Considering these evidences hypothesis, to evaluate the probability of playing tennis or not, we have to calculate the expressions:

```
argmax{P(Yes | Outlook = Sunny, Temperature = Cool, Humidity = High , Wind = Strong),
P(No | Outlook = Sunny, Temperature = Cool, Humidity = High , Wind = Strong)}
```

Calculating each argument separetely we have:

```
P(Yes | (Sunny, Cool, High , Strong)) = P(Yes)*[P(Sunny|Yes)*P(Cool|Yes)*P(High|Yes)*P(Strong|Yes)]
```

And

```
P(No | (Sunny, Cool, High , Strong)) = P(No)*[P(Sunny|No)*P(Cool|No)*P(High|No)*P(Strong|No)]
```

Which gives us:

```
P(Yes | (Sunny, Cool, High , Strong)) = (9/14)*(2/9 * 3/9 * 3/9 * 3/9)
P(No | (Sunny, Cool, High , Strong)) = (5/14)*(3/5 * 1/5 * 4/5 * 3/5)
```

Naming P(Yes | (Sunny, Cool, High , Strong)) P_Yes and P(No | (Sunny, Cool, High , Strong)) P_No and calculating, we have:

```
P_Yes <- (9/14)*(2/9 * 3/9 * 3/9 * 3/9)
P_Yes
```

```
## [1] 0.005291005
```

```
P_No <- (5/14)*(3/5 * 1/5 * 4/5 * 3/5)
P_No
```

```
## [1] 0.02057143
```

As P_No is greater than P_Yes, we will not play tennis tomorrow.