Data Project

Phillip Sparrow

2023-05-28

Part 1

```
# For a single random draw from the deck, what is the probability of selecting a card that is
a club? What is the probability of selecting a card that is 7 or higher?

# Probability of selecting a card that is a club, there are only 4 suits, 1/4
prob_club <- 13/52
prob_club
```

```
## [1] 0.25
```

```
# Probability of selecting a card that is 7 or higher there are 28 cards that are higher than
7
prob_7_or_higher <- 28/52
prob_7_or_higher
```

```
## [1] 0.5384615
```

```r
# A custom function called card.draw to simulate the random drawing of cards with replacement
from a standard 52-card deck of playing cards.

card.draw <- function(n = 1, seed = 0) {
  # Error checking, Check if n is a positive integer (numeric)
  if (!is.numeric(n) || length(n) != 1 || n <= 0) {
    stop("n must be a positive integer.")
  }

  # Error checking, Check if seed is a numeric value
  if (!is.numeric(seed) || length(seed) != 1) {
     stop("seed must be a numeric value.")
  }

  # Set the seed for random number generation
  set.seed(seed)

  # Define suits and card numbers
  suits <- c("C", "D", "H", "S")
  card_numbers <- 1:13

  # Perform random draws
  suit_draws <- sample(suits, n, replace = TRUE)
  number_draws <- sample(card_numbers, n, replace = TRUE)

  # Create the data frame to return the draw
  draws <- data.frame(Suit = suit_draws, Number = number_draws)
  return(draws)
}

# Given example: Draw 15 cards with seed 0
card.draw(n = 15, seed = 0)
```

```
##    Suit Number
## 1     D      5
## 2     C     10
## 3     S      6
## 4     H     10
## 5     C      7
## 6     D      9
## 7     C      5
## 8     H      5
## 9     H      9
## 10    D      9
## 11    D      5
## 12    H      5
## 13    H      2
## 14    C     10
## 15    C      9
```

```r
# Show the output of other scenarios.

card.draw(n = 3.7, seed = 2)
```

```
##   Suit Number
## 1    C      6
## 2    H      8
## 3    D      1
```

```r
card.draw(n = 3, seed = 'a')
```

```
## Error in card.draw(n = 3, seed = "a"): seed must be a numeric value.
```

```r
card.draw(n = c(3, 2), seed = c(1, 2))
```

```
## Error in card.draw(n = c(3, 2), seed = c(1, 2)): n must be a positive integer.
```

```r
card.draw(n = 4, seed = 0.3)
```

```
##   Suit Number
## 1    D      1
## 2    C      2
## 3    S     13
## 4    H      7
```

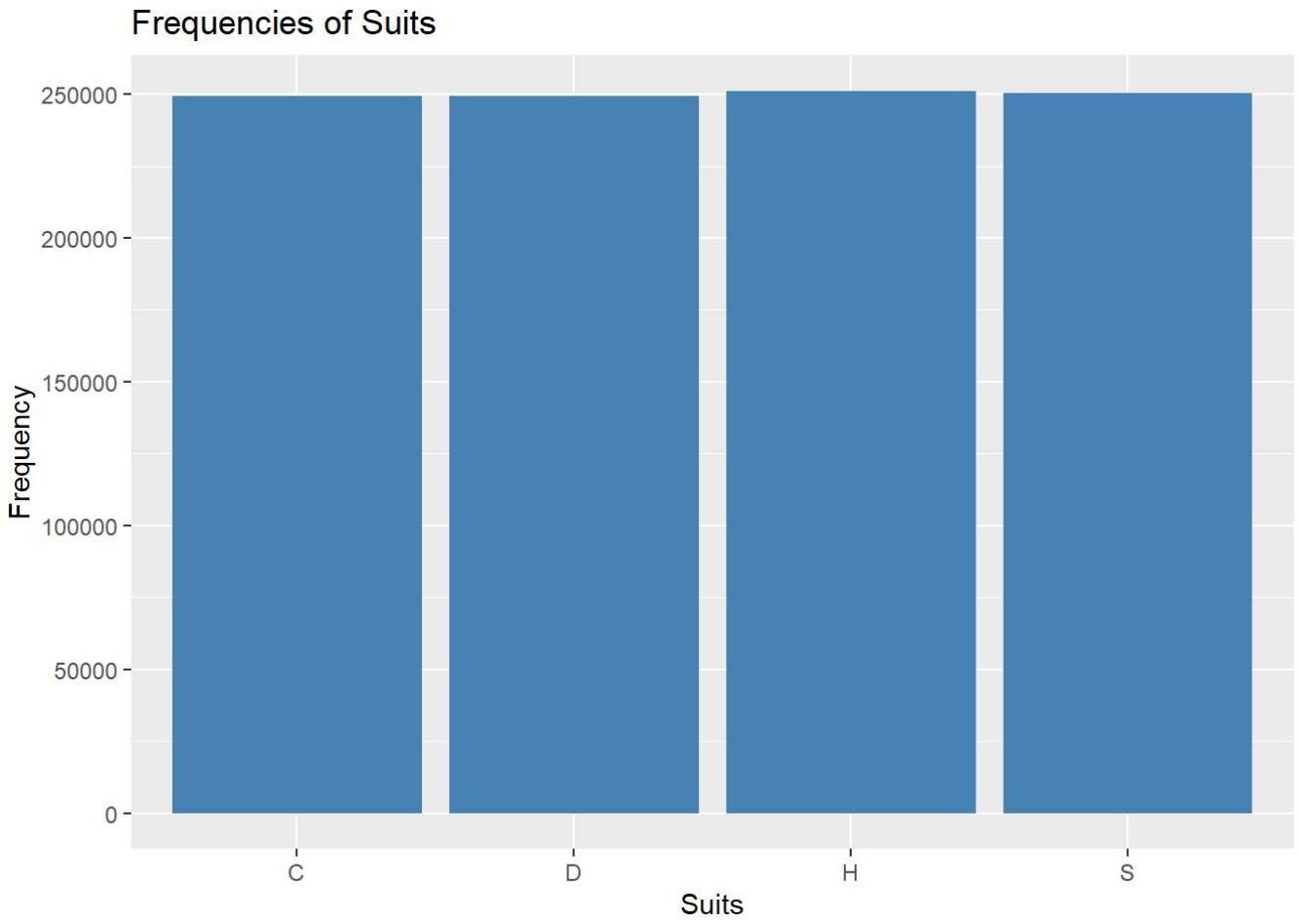
```r
# Simulate 1,000,000 random card draws. Using ggplot to produce appropriate graphical display
s to show frequencies (i.e., the distribution) of the outcomes for:
# the four suits
# the number outcomes for each card.

library(ggplot2)
```

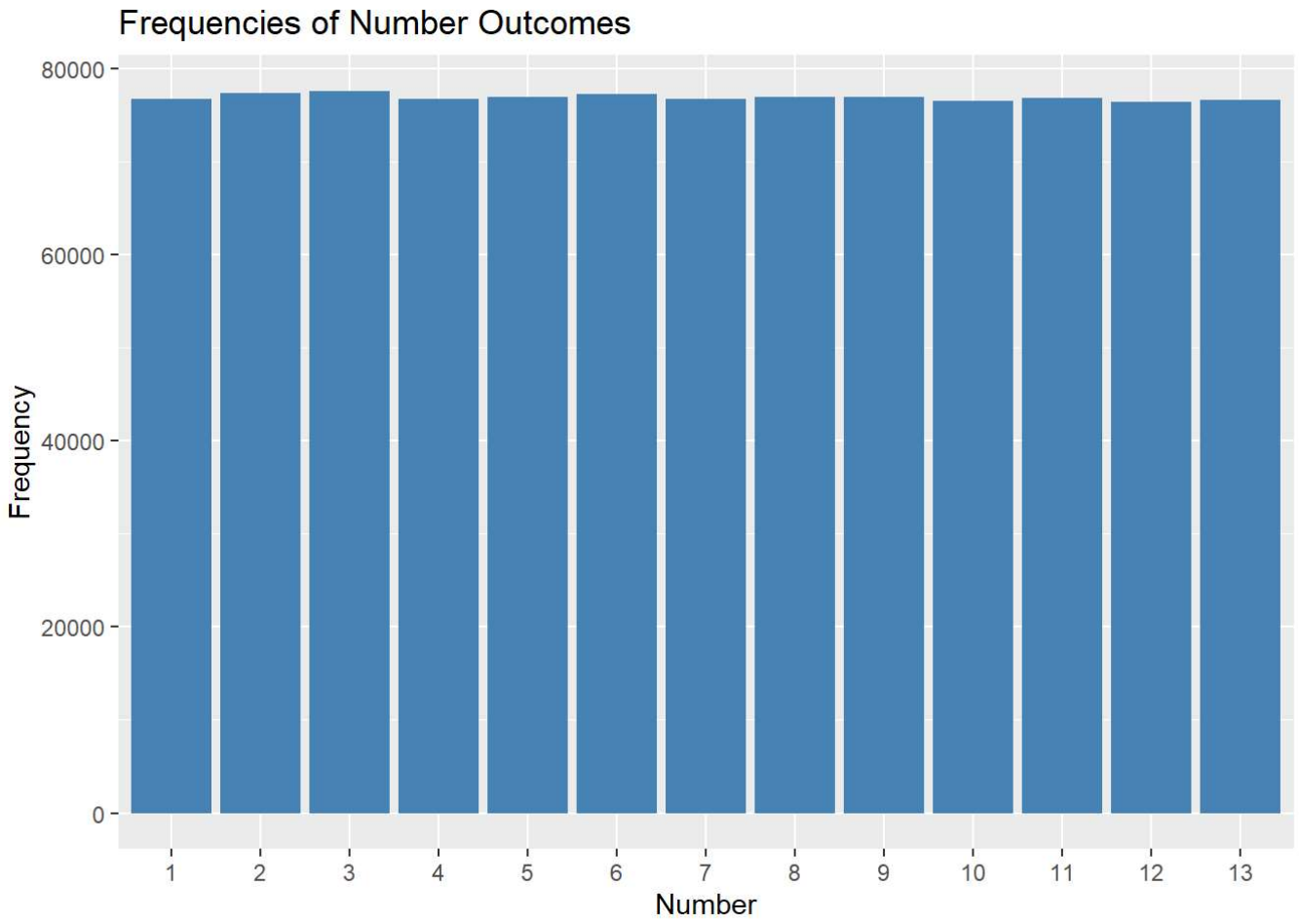
```
## Warning: package 'ggplot2' was built under R version 4.2.3
```

```r
# Simulate 1,000,000 random card draws
draws <- card.draw(n = 1000000)

# Plot the frequencies of the four suits
suit_freq <- table(draws$Suit)
suit_plot <- ggplot(data = data.frame(Suit = names(suit_freq), Frequency = as.numeric(suit_fr
eq)),
                    aes(x = Suit, y = Frequency)) +
                    geom_bar(stat = "identity", fill = "steelblue") +
                    xlab("Suits") +
                    ylab("Frequency") +
                    ggtitle("Frequencies of Suits")
suit_plot
```




```
# Plot the frequencies of the number outcomes for cards
# make graph look more readable by sorting x axis
number_freq <- table(draws$Number)
number_plot <- ggplot(data = data.frame(Number = as.character(names(number_freq)), Frequency
= as.numeric(number_freq)), aes(x = factor(Number, levels = as.character(sort(unique(draws$Nu
mber)))) , y = Frequency)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  xlab("Number") +
  ylab("Frequency") +
  ggtitle("Frequencies of Number Outcomes")
number_plot
```




```
#Simulate 1,000,000 random card draws, and calculate the proportion of the cards drawn that a
re:
#clubs
#7 or higher.

# Simulate 1,000,000 random card draws
draws <- card.draw(n = 1000000)

# Calculate the proportion of cards drawn that are clubs
proportion_clubs <- sum(draws$Suit == "C") / nrow(draws)

# Calculate the proportion of cards drawn that are 7 or higher
proportion_7_higher <- sum(draws$Number >= 7) / nrow(draws)

proportion_clubs
```

```
## [1] 0.249187
```

```
proportion_7_higher
```

```
## [1] 0.53729
```

```
#Exact expected value and variance for the number of outcome for a randomly drawn card from t
he deck (i.e., treating the card drawn as if it were numeric). Using a simulation of 1,000,00
0 random card draws to estimate the expected value and variance to verify the exact values ca
lculated.

# Calculate the exact expected value
exact_expected_value <- sum(1:13 * (4/52))

# Calculate the exact variance
exact_variance <- sum((1:13 - exact_expected_value)^2 * (4/52))

# Simulate 1,000,000 random card draws
random_draws <- card.draw(n = 1000000)

# Calculate the estimated expected value and variance from the simulation
estimated_expected_value <- mean(random_draws$Number)
estimated_variance <- var(random_draws$Number)

# Print the exact expected value and variance
cat("Exact Expected Value:", exact_expected_value, "\n")
```

```
## Exact Expected Value: 7
```

```
cat("Exact Variance:", exact_variance, "\n")
```

```
## Exact Variance: 14
```

```
# Print the estimated expected value and variance
cat("Estimated Expected Value:", estimated_expected_value, "\n")
```

```
## Estimated Expected Value: 6.99156
```

```
cat("Estimated Variance:", estimated_variance, "\n")
```

```
## Estimated Variance: 13.98979
```

Part 2

```r
# Altered Monty Hall Problem. including the possibility that Monty could open the door with t
he car behind it by accident, in this case the player will neither loose or win.

num_iterations <- 10000
iterations_won <- 0; # games won not skipped
wins_stay <- 0
wins_change <- 0

for (i in 1:num_iterations) {    # do 10,000 iterations
  # Randomly assign the door with the car behind it
  car_door <- sample(1:3, 1)

  # Contestant's first choice
  initial_choice <- sample(1:3, 1)

  # Monty opens a door, including the possibility of showing the car
  possible_opened_doors <- 1:3
  # open a door that is not the Contestant's one
  possible_opened_doors <- possible_opened_doors[possible_opened_doors != initial_choice]
  opened_door <- sample(possible_opened_doors, 1)

  # Replay the game if Monty accidentally opens the door with the car
  if (opened_door == car_door) {
    next
  }

  # Contestant's strategy: Stick with the initial choice
  if (initial_choice == car_door) {
    wins_stay <- wins_stay + 1
  }

  # Contestant's strategy: Change doors, take away the first door and the opened door
  new_choice <- setdiff(1:3, c(initial_choice, opened_door))
  if (new_choice == car_door) {
    wins_change <- wins_change + 1
  }
  iterations_won <- iterations_won + 1
}
proportion_wins_stay <- wins_stay / iterations_won
proportion_wins_change <- wins_change / iterations_won
# it seems that the probabilities are roughly the same
proportion_wins_stay   # wins where Contestant stayed
```

```
## [1] 0.5033358
```

```r
proportion_wins_change # wins where Contestant changed
```

```
## [1] 0.4966642
```

Part 3

```r
# Casino type game. A player rolls a standard six-sided die. If the outcome of the roll is a 6, the player loses. If the player rolls any number other than a 6, the player is given up to six additional attempts to roll that number again. The game ends when:
# the player rolls the number again,
# the player rolls a 6, or
# the player has exhausted the six additional rolls.

# Estimating the probability that the player wins with 10,000 playes of the game.
num_iterations <- 10000
wins <- 0

for (i in 1:num_iterations) {
  roll <- sample(1:6, 1)  # Initial roll

  if (roll != 6) {
    # Additional attempts to roll the number again
    for (j in 1:6) {
      roll_again <- sample(1:6, 1)
      if (roll_again == roll) {     # Player wins
        wins <- wins + 1
        break
      } else if (roll_again == 6) { # PLayer rolled a 6, loss
        break
      }
    }                               # player rolled 6 attempts, loss
  }
}

prob_wins <- wins / num_iterations
prob_wins
```

```
## [1] 0.3766
```

Part 4

The following proximity network contains information on whether the actors Allison, Bob, Cass, Dennis, Elliot, and Francis live close to each other:

Allison lives close to Francis.

Allison lives close to Cass.

Bob lives close to Dennis.

Elliot lives close to Cass.

Elliot lives close to Francis.

Francis lives close to Cass.

What type of network, Social network or a bipartite network?

This is a social network, a social network represents relationships between entities. In this case, the given information describes the proximity or closeness between different actors (Allison, Bob, Cass, Dennis, Elliot, and Francis). Whereas a bipartite network is a type of network where the nodes can be divided into two distinct sets, and edges only exist between nodes in different sets.

(b) Directed or undirected?

This is a undirected network, in an undirected network the connections or edges between nodes do not have a specific direction. It means that if node A is connected to node B, therfore node B is also connected to node A.

```
# A network object called proximity.net which reflects the proximity network and which has th
e names of the actors appropriately reflected in the vertex.names node attribute. Output the
adjacency matrix for this network object.
library(network)
```

```
## Warning: package 'network' was built under R version 4.2.3
```

```
##
## 'network' 1.18.1 (2023-01-24), part of the Statnet Project
## * 'news(package="network")' for changes since last version
## * 'citation("network")' for citation information
## * 'https://statnet.org' for help, support, and other information
```

```
# Define the edges representing the proximity relationships
edges <- matrix(c("Allison", "Francis",
                  "Allison", "Cass",
                  "Bob", "Dennis",
                  "Elliot", "Cass",
                  "Elliot", "Francis",
                  "Francis", "Cass"),
                ncol = 2, byrow = TRUE)

# Create the network object
proximity.net <- network(edges, directed = FALSE)

# Set the actor names as vertex attributes
proximity.net %v% "vertex.names" <- c("Allison", "Bob", "Cass", "Dennis", "Elliot", "Franci
s")

# Output the adjacency matrix
as.matrix(proximity.net)
```

```
##         Allison Bob Cass Dennis Elliot Francis
## Allison       0   0    1      0      0       1
## Bob           0   0    0      1      0       0
## Cass          1   0    0      0      1       1
## Dennis        0   1    0      0      0       0
## Elliot        0   0    1      0      0       1
## Francis       1   0    1      0      1       0
```

```
# Are Bob and Cass adjacent?
# Check if Bob and Cass are adjacent
# Convert adjacency matrix to binary matrix
adj_matrix <- as.matrix(proximity.net) > 0

# Check if Bob and Cass are adjacent
is_adjacent <- adj_matrix["Bob", "Cass"]
is_adjacent
```

```
## [1] FALSE
```

```
# Plot the network with node labels
plot(proximity.net, displaylabels = TRUE)
```