Experiment 1 - Dice, Deck of Cards, Slot Machine

Saha Debanshee Gopal

26th August 2016

A) DICE

1) Problem Statement: To plot x and y using qplot

Code:

```
x <- c(-1, -0.8, -0.6, -0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8, 1)
x
```

```
##  [1] -1.0 -0.8 -0.6 -0.4 -0.2  0.0  0.2  0.4  0.6  0.8  1.0
```

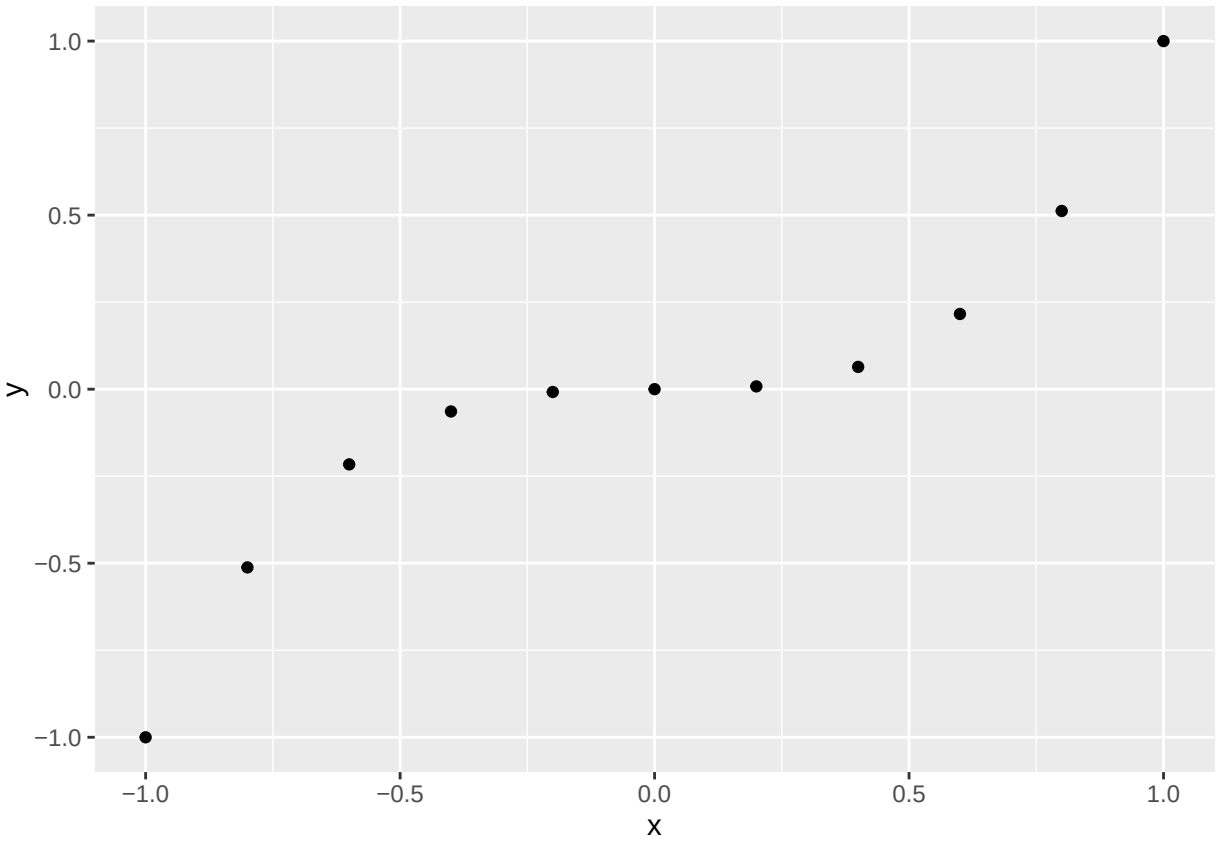
```
y <- x^3
y
```

```
##  [1] -1.000 -0.512 -0.216 -0.064 -0.008  0.000  0.008  0.064  0.216  0.512
## [11]  1.000
```

Here, using th basic computation of R, the cube of x is stored as the value of y.

Output:

```
library(ggplot2)
qplot(x, y)
```



A scatterplot is a set of points, each plotted according to its x and y values. Together, the vectors x and y describe a set of 10 points. Scatterplots are useful for visualizing the relationship between two variables.

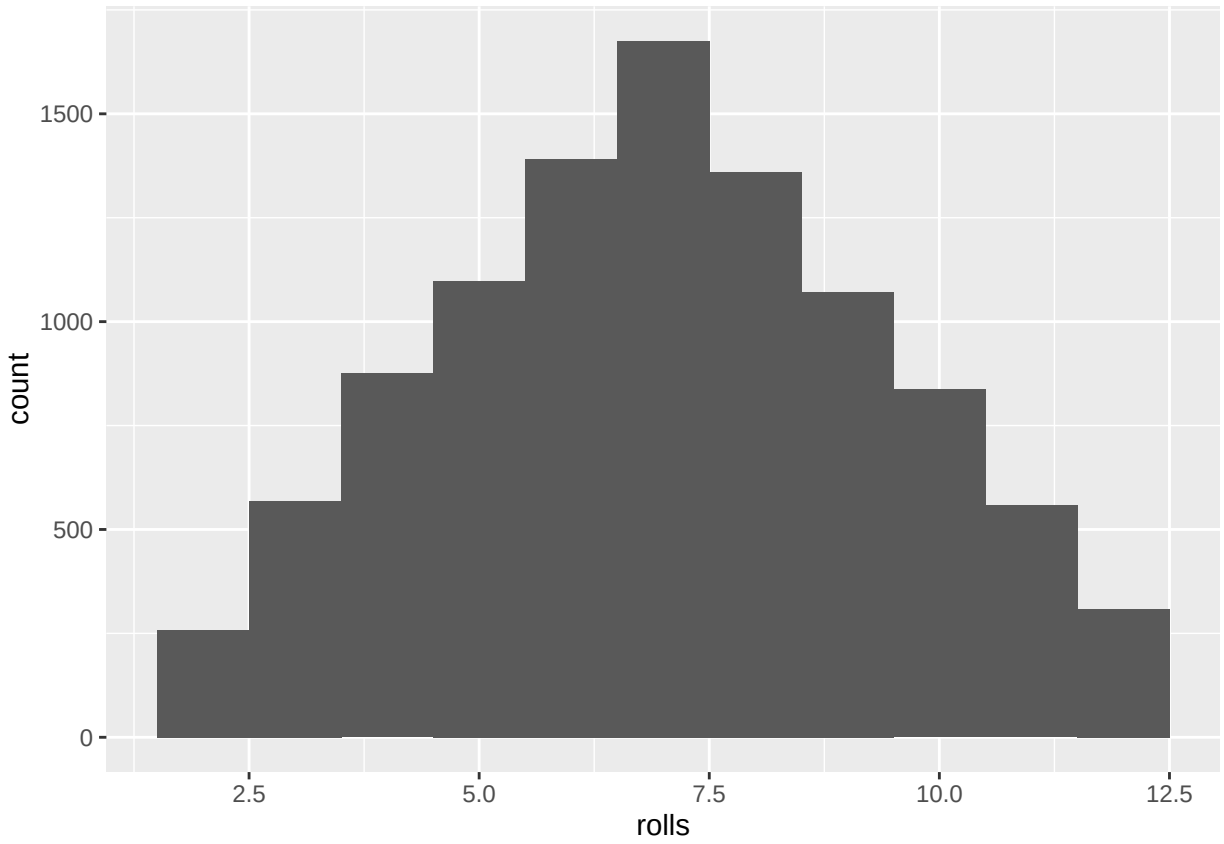
2) Problem Statement: To plot the probability of a normal dice:

Code:

```
roll<-function(){
die<-1:6
dice<-sample(die,size=2,replace=TRUE)
sum(dice)
}
rolls<-replicate(10000,roll())
```

Output:

```
library(ggplot2)
qplot(rolls,binwidth=1)
```



A histogram visualizes the distribution of a single variable. It displays how many data points appear at each value of x. The behavior of our dice suggests that they are fair. Seven occurs more often than any other number, and frequencies diminish in proportion to the number of die combinations that create each number

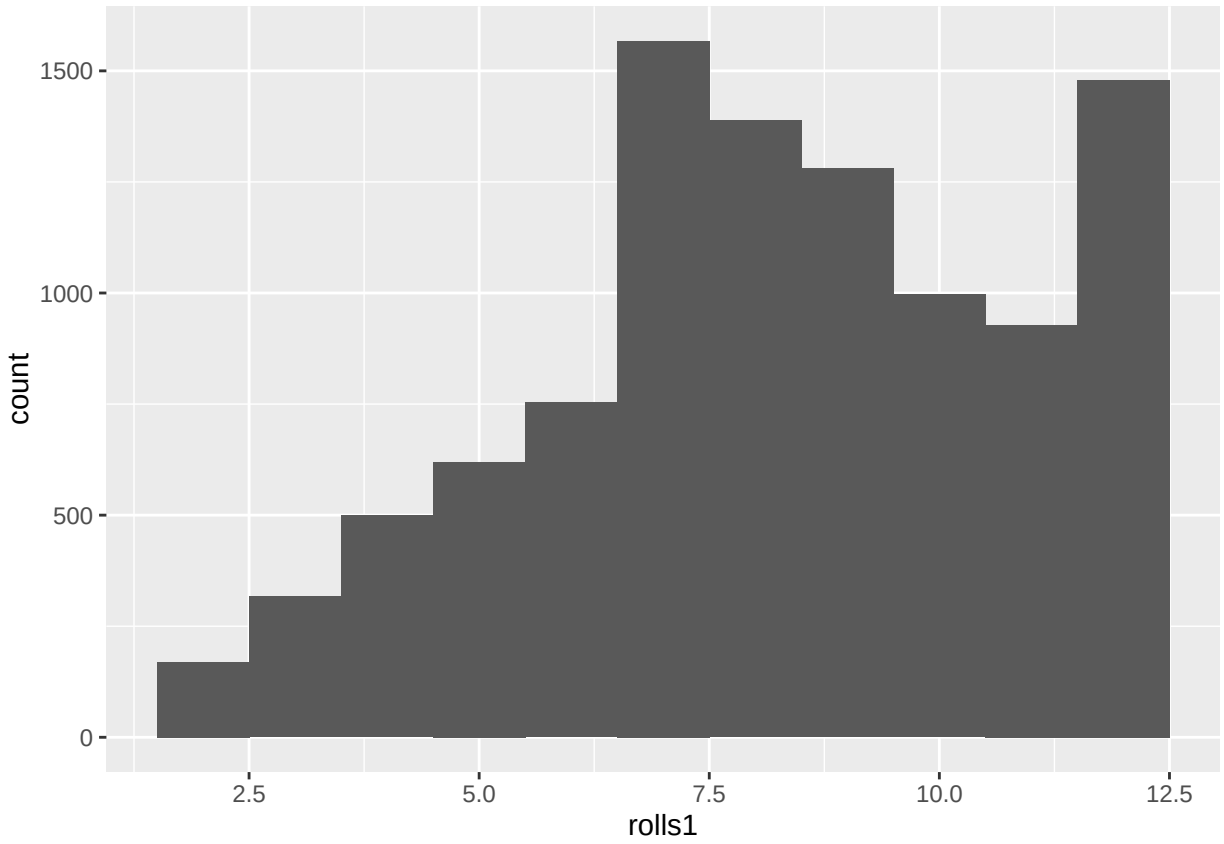
3) Problem Statement: To plot the probability of a weighted dice:

Code:

```
roll1<-function(){
die<-1:6
dice<-sample(die,size=2,replace=TRUE,
prob = c(1/8, 1/8, 1/8, 1/8, 1/8, 3/8))
sum(dice)
}
rolls1<-replicate(10000,roll1())
```

Output:

```
library(ggplot2)
qplot(rolls1,binwidth=1)
```



The dice are now clearly biased towards high numbers, since high sums occur much more often than low sums

B) DECK OF CARDS

1) Problem Statement: Define suits, cards, values

Code

```
suits <- c("Diamonds", "Clubs", "Hearts", "Spades")
face <- c("Ace", "Deuce", "Three", "Four","Five", "Six", "Seven", "Eight",
          "Nine", "Ten", "Jack", "Queen", "King")
values <- c(rep(1:13,4))
```

2) Problem Statement: To show the deck of cards along with its value using data.frame

Code:

```
deck <- data.frame(
face = c("king", "queen", "jack", "ten", "nine", "eight", "seven", "six",
"five", "four", "three", "two", "ace", "king", "queen", "jack", "ten",
"nine", "eight", "seven", "six", "five", "four", "three", "two", "ace",
"king", "queen", "jack", "ten", "nine", "eight", "seven", "six", "five",
"four", "three", "two", "ace", "king", "queen", "jack", "ten", "nine",
"eight", "seven", "six", "five", "four", "three", "two", "ace"),
suit = c("spades", "spades", "spades", "spades", "spades", "spades",
"spades", "spades", "spades", "spades", "spades", "spades", "spades",
```

```
"clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs",
"clubs", "clubs", "clubs", "clubs", "clubs", "diamonds", "diamonds",
"diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "diamonds",
"diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "hearts",
"hearts", "hearts", "hearts", "hearts", "hearts", "hearts", "hearts",
"hearts", "hearts", "hearts", "hearts", "hearts"),
value = c(13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8,
7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11,
10, 9, 8, 7, 6, 5, 4, 3, 2, 1))
```

Output:

```
deck
```

```
##     face     suit value
## 1   king   spades    13
## 2  queen   spades    12
## 3   jack   spades    11
## 4    ten   spades    10
## 5   nine   spades     9
## 6  eight   spades     8
## 7  seven   spades     7
## 8    six   spades     6
## 9   five   spades     5
## 10  four   spades     4
## 11 three   spades     3
## 12   two   spades     2
## 13   ace   spades     1
## 14  king    clubs    13
## 15 queen    clubs    12
## 16  jack    clubs    11
## 17   ten    clubs    10
## 18  nine    clubs     9
## 19 eight    clubs     8
## 20 seven    clubs     7
## 21   six    clubs     6
## 22  five    clubs     5
## 23  four    clubs     4
## 24 three    clubs     3
## 25   two    clubs     2
## 26   ace    clubs     1
## 27  king diamonds    13
## 28 queen diamonds    12
## 29  jack diamonds    11
## 30   ten diamonds    10
## 31  nine diamonds     9
## 32 eight diamonds     8
## 33 seven diamonds     7
## 34   six diamonds     6
## 35  five diamonds     5
## 36  four diamonds     4
## 37 three diamonds     3
## 38   two diamonds     2
```

```
## 39   ace diamonds     1
## 40  king   hearts    13
## 41 queen   hearts    12
## 42  jack   hearts    11
## 43   ten   hearts    10
## 44  nine   hearts     9
## 45 eight   hearts     8
## 46 seven   hearts     7
## 47   six   hearts     6
## 48  five   hearts     5
## 49  four   hearts     4
## 50 three   hearts     3
## 51   two   hearts     2
## 52   ace   hearts     1
```

3) Problem Statement: To shuffle and deal a deck

Code:

```
deal <- function(cards) {
cards[1, ]
}
shuffle <- function(cards) {
random <- sample(1:52, size = 52)
cards[random, ]
}
```

Output:

```
deal(deck)
```

```
##   face   suit value
## 1 king spades    13
```

```
deck2 <- shuffle(deck)
```

Output 2:

```
deal(deck2)
```

```
##     face     suit value
## 28 queen diamonds    12
```

4) Problem Statement: Creating Deck

Code:

```
suits <- c("Diamonds", "Clubs", "Hearts", "Spades")
face <- c("Ace", "Deuce", "Three", "Four","Five", "Six", "Seven", "Eight",
          "Nine", "Ten", "Jack", "Queen", "King")
values <- c(rep(1:13,4))
```

```
deck <- expand.grid(face=face, suits=suits)
deck$value <- values


deck2 <- deck

deck2$new <- 1:52

deck2[c(13, 26, 39, 52), ]
```

```
##    face    suits value new
## 13 King Diamonds    13  13
## 26 King    Clubs    13  26
## 39 King   Hearts    13  39
## 52 King   Spades    13  52
```

5) Problem Statement: Count the number of Ace in Deck

Code:

```
deck2$face == "Ace"
```

```
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [34] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
## [45] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
sum(deck2$face == "Ace")
```

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

6) Problem Statement: Assign value 14 to every Ace in deck3

Code

```
deck3 <- deck
deck3$value[deck3$face == "Ace"] <- 14
```

7) Problem Statement: Creating Deck for Hearts

Code

```
deck4<-deck

deck4$value <- 0
deck4$value[deck4$suits == "Hearts"] <- 1
deck4$value[deck4$suits == "Hearts"]
```

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

```
deck4$face == "Queen" & deck4$suits == "Spades"
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [12] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [23] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [34] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [45] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
```

```
queenofsp <- deck4$face == "Queen" & deck4$suits == "Spades"
deck4[queenofsp, ]
```

```
##     face  suits value
## 51 Queen Spades     0
```

```
deck4$value[queenofsp] <- 13
deck4$face[queenofsp] <- "Queen"
deck4$suits[queenofsp] <- "Spades"
```

Output:

```
knitr::kable(deck4)
```

face	suits	value
Ace	Diamonds	0
Deuce	Diamonds	0
Three	Diamonds	0
Four	Diamonds	0
Five	Diamonds	0
Six	Diamonds	0
Seven	Diamonds	0
Eight	Diamonds	0
Nine	Diamonds	0
Ten	Diamonds	0
Jack	Diamonds	0
Queen	Diamonds	0
King	Diamonds	0
Ace	Clubs	0
Deuce	Clubs	0
Three	Clubs	0
Four	Clubs	0
Five	Clubs	0
Six	Clubs	0
Seven	Clubs	0
Eight	Clubs	0
Nine	Clubs	0
Ten	Clubs	0
Jack	Clubs	0
Queen	Clubs	0
King	Clubs	0
Ace	Hearts	1

face	suits	value
Deuce	Hearts	1
Three	Hearts	1
Four	Hearts	1
Five	Hearts	1
Six	Hearts	1
Seven	Hearts	1
Eight	Hearts	1
Nine	Hearts	1
Ten	Hearts	1
Jack	Hearts	1
Queen	Hearts	1
King	Hearts	1
Ace	Spades	0
Deuce	Spades	0
Three	Spades	0
Four	Spades	0
Five	Spades	0
Six	Spades	0
Seven	Spades	0
Eight	Spades	0
Nine	Spades	0
Ten	Spades	0
Jack	Spades	0
Queen	Spades	13
King	Spades	0

8) Problem Statement: Keeping original deck safe (Closure)

Code:

```
setup <- function(deck) {
DECK <- deck
DEAL <- function() {
card <- deck[1, ]
assign("deck", deck[-1, ], envir = parent.env(environment()))
card
}
SHUFFLE <- function(){
random <- sample(1:52, size = 52)
assign("deck", DECK[random, ], envir = parent.env(environment()))
}
list(deal = DEAL, shuffle = SHUFFLE)
}
cards <- setup(deck)
deal <- cards$deal
shuffle <- cards$shuffle
```

Closure ensures that even if we remove the original deck, we can continue playing cards.

C) SLOT MACHINE

A code in R which allows us to play the most popular mordern casino game.

Symbols used include the following:

DD - Diamonds (0.03)

7 - Seven (0.03)

BBB - Triple Bars (0.06)

BB - Double Bars (0.1)

B - Single Bars (0.25)

C - Cherries (0.01)

0 - Zeros (0.52)

with the probabilities in the brackets.

1) Problem Statement: To select symbols randomly using the sample function

Code

```
get_symbols <- function() {
  wheel <- c("DD","7","BBB","BB","B","C","0")
  sample(wheel,size = 3,replace= TRUE,prob = c(0.03,0.03,0.06,0.1,0.25,0.01,0.52))
}
```

Output:

```
get_symbols()
```

```
## [1] "0" "B" "0"
```

For the actual game, we need to assign score to the symbols and that can be done via the score function. The score of the 3 random symbols obtained from 'get_symbols' is extracted from a lookup table which has all the values.

Code:

```
score <- function(symbols){
  same <- symbols[1] == symbols[2] && symbols[2] == symbols[3]
  bars <- symbols %in% c("B","BB","BBB")
  if(same){
    payouts <- c("DD" = 100,
                 "7" = 80,
                 "BBB" = 40,
                 "BB" = 25,
                 "B" = 10,
                 "C" = 10,
                 "0" = 0)
    prize <- unname(payouts[symbols[1]])
  }
  else if(all(bars)){
    prize <- 5
```

```
  }
  else{
    cherries <- sum(symbols == "C")
    prize <- c(0,2,5)[cherries +1]
  }

  diamonds <- sum(symbols == "DD")
  prize * 2 ^ diamonds
}
```

The game can be run using this function which calls the get_symbols() functions and the score() function.

Code:

```
play <- function(){
  symbols <- get_symbols()
  print(symbols)
  score(symbols)
}
play()
```

```
## [1] "0" "0" "0"
```

```
## [1] 0
```

Modified play() function to store the values of the symbols as attributes with the value of prize.

Code:

```
play <- function(){
  symbols <- get_symbols()
  structure(score(symbols),symbols = symbols)
}
play()
```

```
## [1] 0
## attr(,"symbols")
## [1] "0" "B" "0"
```

The structure function creates an object with a set of attributes. The first argument should be a R object or set of values and the remaining arguments should be named attributes for the structure to add to the object.

The attributes now can be used to create a slot_display() functiona as follows:

```
slot_display <- function(prize){
  #extract the symbols
  symbols <- attr(prize,"symbols")
  #combine symbol with prize as a regular expression
  symbols <- paste(symbols,collapse = " ")
  #append with new line character
  string <- paste(symbols,prize,sep="\n$")
  cat(string)   #display without quotes
}
one_play <- play()
```

We use expand.grid to find out all the possible combinations of a vector with another vector. Using this we calculate the possible combinations of the wheel.

```
wheel <- c("DD","7","BBB","BB","B","C","0")
combos <- expand.grid(wheel,wheel,wheel,stringsAsFactors = FALSE)
head(combos,3)
```

```
##   Var1 Var2 Var3
## 1   DD   DD   DD
## 2    7   DD   DD
## 3  BBB   DD   DD
```

This creates a variable combos with 343 observations.

We then create a new lookup table for the probabilites and add those values to combos as a factor and, calculate and add a total probability for each of the combination.

```
prob <- c("DD" = 0.03,"7" = 0.03,"BBB" = 0.06,"BB" = 0.1,"B" = 0.25,"C" = 0.01,"0" = 0.52)

combos$prob1 <- prob[combos$Var1]
combos$prob2 <- prob[combos$Var2]
combos$prob3 <- prob[combos$Var3]

combos$prob = combos$prob1 * combos$prob2 * combos$prob3
head(combos,3)
```

```
##   Var1 Var2 Var3 prob1 prob2 prob3    prob
## 1   DD   DD   DD  0.03  0.03  0.03 2.7e-05
## 2    7   DD   DD  0.03  0.03  0.03 2.7e-05
## 3  BBB   DD   DD  0.06  0.03  0.03 5.4e-05
```

To store the prize along with the combinations, we use a for loop and add prize as a factor.

```
combos$prize <- NA
for(i in 1:nrow(combos)){
  symbols <- c(combos[i,1],combos[i,2],combos[i,3])
  combos$prize[i] <- score(symbols)
}
head(combos,3)
```

```
##   Var1 Var2 Var3 prob1 prob2 prob3    prob prize
## 1   DD   DD   DD  0.03  0.03  0.03 2.7e-05   800
## 2    7   DD   DD  0.03  0.03  0.03 2.7e-05     0
## 3  BBB   DD   DD  0.06  0.03  0.03 5.4e-05     0
```

The expected value of the prize won is given by

```
sum(combos$prize * combos$prob)
```

```
## [1] 0.538014
```

This value is less than the value mentioned by the manufacturer because of the wild card "DD".

Correcting the code to fix the wild card problem and recalculating the expected value of prize.

```
score <- function(symbols){
  diamonds <- sum(symbols == "DD")
  cherries <- sum(symbols == "C")
  #case identification
  slots <- symbols[symbols != "DD"]
  same <- length(unique(slots)) == 1
  bars <- slots %in% c("B","BB","BBB")
  #assign prize
  if(diamonds == 3){
    prize <- 100
  } else if(same){
    payouts <- c("7"=80,"BBB"=40,"BB"=25,"B"=10,"C"=10,"0"=0)
    prize <- unname(payouts[slots[1]])
  } else if(all(bars)){
    prize <- 5
  } else if(cherries > 0){
    prize <- c(0,2,5)[cherries + diamonds + 1]
  } else {
    prize <- 0
  }
  #double the prize for each diamond
  prize * 2^diamonds
}

#reassinging the prize value
combos$prize <- NA
for(i in 1:nrow(combos)){
  symbols <- c(combos[i,1],combos[i,2],combos[i,3])
  combos$prize[i] <- score(symbols)
}

#finding sum
sum(combos$prize * combos$prob)
```

```
## [1] 0.934356
```

2) Problem Statement: To demonstrate the playing in till the cash runs out

Code:

```
plays_till_broke <- function(start_with) {
  cash <- start_with
  n<-0
  while(cash >0){
    cash <- cash - 1 + play()
    n <- n + 1
  }
  n
}
```