R Programming Example: Create a dice-rolling program

Mary Amon

2023-03-15

```
set.seed(123) # for reproducibility
```

Adapted from code examples in Hands-On Programming with R by Garrett Grolemund

In this example, we will

- 1. Create a 6-sided die object
- 2. Create a function that will roll any number of 6-sided dice
- 3. Use the function within a for loop to roll the dice 100 times
- 4. Create a visualization of the 100 dice rolls

Note that this example only uses the Base R language without any imported packages. In practice, you would import specialized packages to manage data wrangling and data visualization.

1. Create a 6-sided die

The : operator will create an atomic vector, a one-dimensional set of 1 data type. The <- operator is the assignment operator to save data to an R object.

```
# the operator : to create a 1-dimensional vector with values 1,2,3,4,5,6
# the operator <- assigns the vector to the R object called "die"
die <- 1:6
```

2. Create a function that will roll any number of 6 sided dice

Base R has a function called sample() that we can use to "roll" the die. sample() takes a sample from a set of elements.

Every parameter in every R function has a name. You can specify which data should be assigned to which argument. This is optional, but aids readability and prevents errors as you use functions with many parameters.

To roll 1 die, the parameters are:

- x: the elements to choose from (die)
- size: the number of items to choose (1 roll)

Note: by default, sample() will choose a value from x with equal probability for each element.

```
sample(x=die,size = 1)
```

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

If we want to roll 2 dice, we need to pass an additional parameter called replace.

When we pick a value from [1,2,3,4,5,6] for the first time, we want to put back that value so that when we pick the second value, all the values are still in the set.

To roll 2 dice, the parameters are:

- x: the elements to choose from (die)
- size: the number of items to choose (2 rolls)
- replace: put the value back when selecting the next element (TRUE)

```
sample(die,size = 2, replace = TRUE)
```

```
## [1] 6 3
```

We can put the previous bit of code within a function that will let us roll any number of 6-sided dice we want.

Functions in R are another type of R object. Instead of containing data, they contain code.

Every function in R has 3 basic parts. Here are the parts for our roll function:

- 1. **Name:** use the <- operator to assign the function to the R object roll
- 2. **Body of code:** within curly braces, define the die and call sample() to "roll" dice
- 3. **Parameters:** within parenthesis, add parameter name numDice for how many dice we want to roll

```
roll <- function(numDice){
  die <- 1:6
  sample(die, size=numDice, replace = TRUE)
}
```

Now, we can call our function to play any games that have dice!

```
#Craps have 2 dice
crapsDice = 2
crapsRoll = roll(crapsDice)
print(crapsRoll)
```

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

```
#Yahtzee has 5 dice
yahtzeeDice = 5
yahtzeeRoll = roll(yahtzeeDice)
print(yahtzeeRoll)
```

```
## [1] 6 3 5 4 6
```

3. Use the function within a for loop to roll the dice 100 times

Like other programming languages, R has loops for repeating a task.

for loop in R resembles for loop in other languages, as it uses:

- the keyword "for"
- a parameter indicating how to loop
- a body of code to loop over.

Similar to Python, the parameter for looping is iterating through a list.

To save output our loop to roll dice, we need to save the result as it runs. We will use that in a dataframe, which is a 2-dimensional version of a list. A dataframe groups vectors together in a two-dimensional table. Each column of a data frame can be a different type of data, but within a column, every cell must be the same type.

Note: If you use a for loop in R, you will often run into "Hey, you shouldn't be using a for loop!". That being said, sometimes you just need to use a loop, you know?

```
data <- data.frame(die1=roll(1),die2=roll(1)) #initialize dataframe with 1 roll
for (i in 1:99){ #iterate through the values in the atomic vector 1 to 100
  data[nrow(data) + 1, ] <- roll(crapsDice) # roll the dice and store it in the next row
}
head(data,3) # inspect the table as a sanity check
```

```
##   die1 die2
## 1    6    1
## 2    2    3
## 3    5    3
```

4. Create a visualization of the 100 dice rolls

We can access columns in a dataframe with the $ character. Let's add a new column called "total" adding up the value from die1 and die2

```
data$total <- data$die1 + data$die2
head(data,3)
```

```
##   die1 die2 total
## 1    6    1     7
## 2    2    3     5
## 3    5    3     8
```

We can use the Base R function aggregate() to group all the 1s, 2s, etc.

The parameters of aggregate() are:

- x: the data to be split into groups by grouping
- by: the list of grouping elements
- FUN: a function to compute the summary statistics. Here we use length, which is "length" in the sense of the length of a list. You could also use FUN = mean, median, sum, etc.

```
totalcounts <-aggregate(x=data$total, by = list(data$total), FUN = length)
print(totalcounts)
```

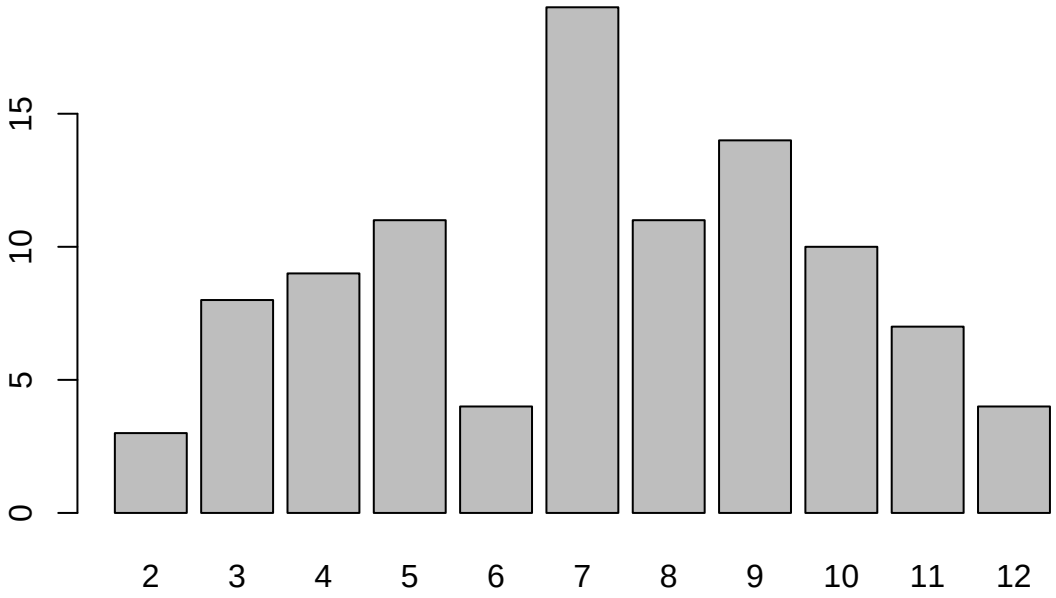
```
##    Group.1  x
## 1        2  3
## 2        3  8
## 3        4  9
## 4        5 11
## 5        6  4
## 6        7 19
## 7        8 11
## 8        9 14
## 9       10 10
## 10      11  7
## 11      12  4
```

We can graph this distribution using Base R barplot() function.

Parameters for barplot():

- height: a vector of values describing the height of each bar in the barplot should be
- names.arg: a vactor of names to be plotted below each bar

```
myplot <- barplot(height=totalcounts$x,names.arg=totalcounts$Group.1)
```




```
myplot
```

```
##       [,1]
##  [1,]  0.7
##  [2,]  1.9
##  [3,]  3.1
##  [4,]  4.3
##  [5,]  5.5
##  [6,]  6.7
##  [7,]  7.9
##  [8,]  9.1
##  [9,] 10.3
## [10,] 11.5
## [11,] 12.7
```

Bonus: Compare our frequency table to the frequency of all possible combinations

Base R has a function called expand.grid that returns a dataframe of all possible combinations.

```
rolls <- expand.grid(die1=die,die2=die)
head(rolls,3)
```

```
##   die1 die2
## 1    1    1
## 2    2    1
## 3    3    1
```

We can access columns in a dataframe with the $ character. Let's add a new column called "total" adding up the value from die1 and die2

```
rolls$total <- rolls$die1 + rolls$die2
head(rolls,3)
```

```
##   die1 die2 total
## 1    1    1     2
## 2    2    1     3
## 3    3    1     4
```

Base R aggregate() to group all the 1s, 2s, etc, in the total

```
freq_df <-aggregate(rolls$total, by = list(rolls$total), FUN = length)
freq_df
```

```
##    Group.1 x
## 1        2 1
## 2        3 2
## 3        4 3
## 4        5 4
## 5        6 5
## 6        7 6
## 7        8 5
## 8        9 4
## 9       10 3
## 10      11 2
## 11      12 1
```

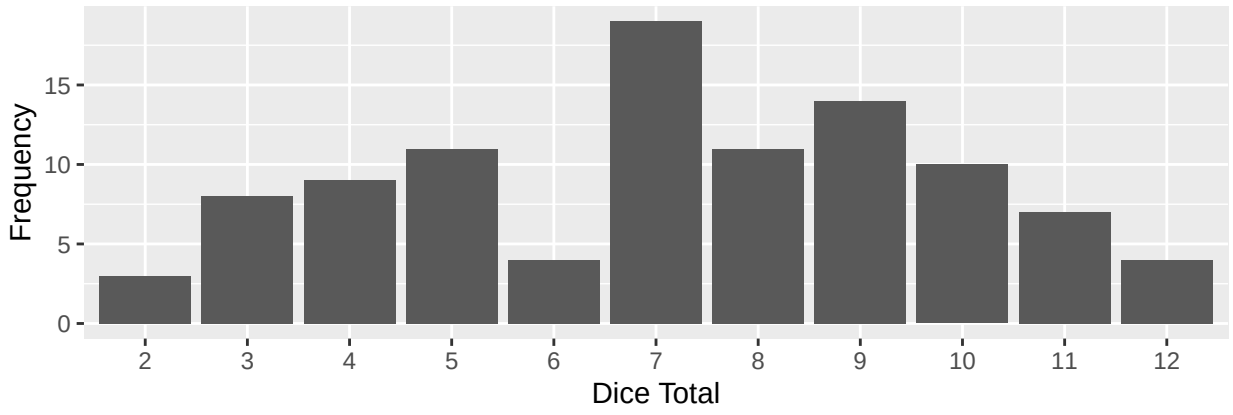
Compare our simulated rolls above to the distribution of all possible combinations.

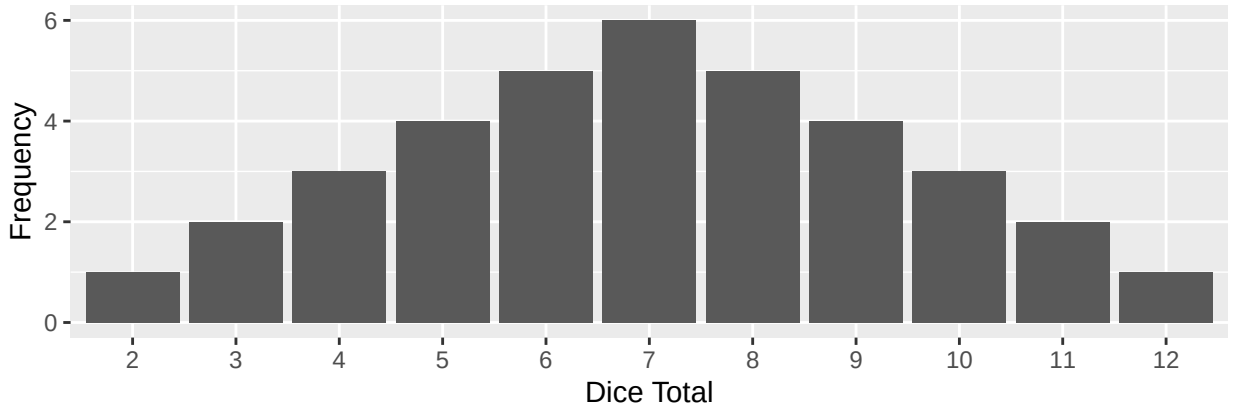
```
library(ggplot2)
library(gridExtra)

totalcounts$Group.1 <- as.factor(totalcounts$Group.1)
myplot2 <- ggplot(data=totalcounts,aes(x=Group.1,y=x)) +
  geom_bar(stat="identity") +
  xlab("Dice Total") +
  ylab("Frequency")

freq_df$Group.1 <- as.factor(freq_df$Group.1)
distplot <- ggplot(data=freq_df,aes(x=Group.1,y=x)) +
  geom_bar(stat="identity") +
  xlab("Dice Total") +
  ylab("Frequency")

grid.arrange(myplot2,distplot)
```


Frequency
15
10
5
0
2
3
4
5
6
7
8
9
10
11
12
Dice Total


Frequency
6
4
2
0
2
3
4
5
6
7
8
9
10
11
12
Dice Total