**Data Science: Probability**

**with Prof. Rafael Irizarry**

**James Strayer**Course Notes and Screenshots

HarvardX  
Data Science: Probability

**Module No.3**

**Data Science: Probability**

with Prof. Rafael Irizarry

**Module 3**

**Table of Contents**

[1. Section 1: Discrete Probability 3](#_Toc17718184)

[1.1. Introduction to Discrete Probability 3](#_Toc17718185)

[1.1.1. Discrete Probability 3](#_Toc17718186)

[1.1.2. Monte Carlo Simulations 4](#_Toc17718187)

[1.1.3. Setting the Random Seed 5](#_Toc17718188)

[1.1.4. Using the mean Function for Probability 6](#_Toc17718189)

[1.1.5. Probability Distributions 7](#_Toc17718190)

[1.1.6. Independence 7](#_Toc17718191)

[1.2. Combinations and Permutations 8](#_Toc17718192)

[1.2.1. Combinations and Permutations 8](#_Toc17718193)

[1.2.2. The Birthday Problem 8](#_Toc17718194)

[1.2.3. sapply 8](#_Toc17718195)

[1.2.4. How Many Monte Carlo Experiments are Enough? 8](#_Toc17718196)

[1.3. Addition Rule and Monty Hall 8](#_Toc17718197)

[1.3.1. The Addition Rule 8](#_Toc17718198)

[1.3.2. The Monty Hall Problem 8](#_Toc17718199)

[2. Section 2: Continuous Probability 9](#_Toc17718200)

[2.1. Continuous Probability 9](#_Toc17718201)

[2.1.1. Continuous Probability 9](#_Toc17718202)

[2.1.2. Theoretical Distribution 9](#_Toc17718203)

[2.1.3. Probability Density 9](#_Toc17718204)

[2.1.4. Monte Carlo Simulations 9](#_Toc17718205)

[2.1.5. Other Continuous Distributions 9](#_Toc17718206)

[3. Section 3: Random Variables, Sampling Models, and the Central Limit Theorem 10](#_Toc17718207)

[3.1. Random Variables and Sampling Models 10](#_Toc17718208)

[3.1.1. Random Variables 10](#_Toc17718209)

[3.1.2. Sampling Models 10](#_Toc17718210)

[3.1.3. Distributions versus Probability Distributions 10](#_Toc17718211)

[3.1.4. Notation for Random Variables 10](#_Toc17718212)

[3.1.5. Central Limit Theorem 10](#_Toc17718213)

[3.2. The Central Limit Theorem Continued 10](#_Toc17718214)

[3.2.1. Averages and Proportions 10](#_Toc17718215)

[3.2.2. Law of Large Numbers 10](#_Toc17718216)

[3.2.3. How Large is Large in CLT? 10](#_Toc17718217)

[4. Section 4: The Big Short 11](#_Toc17718218)

[4.1. The Big Short 11](#_Toc17718219)

[4.1.1. The Big Short: Interest Rates Explained 11](#_Toc17718220)

[4.1.2. The Big Short 11](#_Toc17718221)

**List of Figures**

**Aucune entrée de table d'illustration n'a été trouvée.**

1. Section 1: Discrete Probability
   1. Introduction to Discrete Probability
      1. Discrete Probability

We start by covering some basic principles related to categorical data. This subset of probability is referred to as discrete probability. It will help us understand the probability theory we will later introduce for numeric and continuous data. Discrete probability is more useful in card games and we use these as examples.

The word probability is used in everyday language. For example, Google`s auto complete of “What are the chances of” give us “getting pregnant”, “having twins” …Answering questions about probability is often hard, if not impossible. Here we discuss a mathematical definition of probability that does permit us to give precise answers to certain questions. For example, If I have 2 red beans and 3 blue beans inside an urn and I pick one at random, what is the probability of picking a red one? Our intuition tells us that the answer is 2/5, or 40%. A precise definition can be given by noting that there are 5 possible outcomes of which 2 satisfy the condition necessary for the event “pick a red bead”. Because each of the 5 outcomes has the same chance of occurring, we conclude that the probability is 0.4 for red and 0.6 for blue. A more tangible way to think about the probability of an event is a as proportion of times the event occurs when we repeat the experiment over and over independently and under the same conditions.

Before we continue let’s introduce some notation. We use the notation of :

To denote the probability of an event A happening. For example the event “picking a red bean”. In Data Science applications, we will often deal with continuous variables. In these cases we write event more a in mathematical form. For example the event “a person being taller than 6’ foot” will be written:

)

* + 1. Monte Carlo Simulations

Computer provide a way to perform the simple random experiments, such as the one we did before. Pick a bean at random from a bag or an urn with 3 blue beans and 2 red one. **Random generators** permit us to mimic the process of picking at random. An example in R is the **sample()** function, first we use the **rep()** function to generate the urn:

*beads<-rep(c("red","blue"),times=c(2,3))*

*beads*

*[1] "red" "red" "blue" "blue" "blue"*

Now we can use a sample function to pick one at random:

*sample(beads,1)*

*[1] "blue"*

Now, we want to repeat this experiment over and over. However it is impossible to repeat forever, instead we repeat the experiment a large enough number of times to make the results practically equivalent to doing forever. This is an example of a **Monte Carlo simulation**. To perform our first Monte Carlo simulation we use the **replicate()** function to sample from the bean 10000 times:

*B<-10000*

*events<-replicate(B,sample(beads,1))*

We can use **table()** to see the distribution and we can use **prop.table()** to give us the proportions:

*tab<-table(events)*

*> tab*

*events*

*blue red*

*5911 4089*

*> prop.table(tab)*

*events*

*blue red*

*0.591 0.409*

Note that we don’t need to use replicate in this example. This is because the function sample has an argument that permits us to pick more than one element from the urn. However by default, this selection occurs without replacement. After a bead is selected, it is not put back in the urn. Look what happen when we ask to randomly select 5 beads. Let’s do it 3 times:

*sample(beads,5)*

*[1] "blue" "red" "blue" "red" "blue"*

*> sample(beads,5)*

*[1] "red" "blue" "blue" "red" "blue"*

*> sample(beads,5)*

*[1] "blue" "red" "red" "blue" "blue"*

This results in a rearrangement that always has 3 blue and 2 red beads. If we ask for six bead we get an error. It tells you that you don’t have enough beads to take 6, it’s because it’s doing it without replacement:

*sample(beads,6)*

*Error in sample.int(length(x), size, replace, prob) :*

*cannot take a sample larger than the population when 'replace = FALSE'*

However the sample function can be used directly again, without the replicate to repeat the same experiment of picking 1 out of 5 beads multiple times under the same conditions. To do this, we sample WITH replacement:

*events<-sample(beads,B,replace = TRUE)*

*> prop.table(table(events))*

*events*

*blue red*

*0.59 0.41*

* + 1. Setting the Random Seed

Before we continue, we will briefly explain the following important line of code:

set.seed(1986)

Throughout this book, we use random number generators. This implies that many of the results presented can actually change by chance, which then suggests that a frozen version of the book may show a different result than what you obtain when you try to code as shown in the book. This is fine since the results are random and change from time to time. However, if you want to ensure that results are the same every time you run them, you can set R’s random number generation seed to a specific number. Above we set it to 1986. We want to avoid using the same seed every time. A popular way to pick the seed is the year - month - day. For example, we picked 1986 on December 20, 2018: 2018 − 12 − 20 = 1986.

You can learn more about setting the seed by looking at the documentation:

*?set.seed*

In the exercises, we may ask you to set the seed to assure that the results you obtain are exactly what we expect them to be.

Important note on seeds in R 3.5 and R 3.6. R was recently updated to version 3.6 in early 2019. In this update, the default method for setting the seed changed. This means that exercises, videos, textbook excerpts and other code you encounter online may yield a different result based on your version of R.If you are running R 3.6, you can revert to the original seed setting behavior by adding the argument sample.kind="Rounding". For example:

*set.seed(1)*

*set.seed(1, sample.kind="Rounding") # will make R 3.6 generate a seed as in R 3.5*

*Using the sample.kind="Rounding" argument will generate a message:*

*non-uniform 'Rounding' sampler used*

This is not a warning or a cause for alarm - it is a confirmation that R is using the alternate seed generation method, and you should expect to receive this message in your console.

If you use R 3.6, you should always use the second form of set.seed in this course series (outside of DataCamp assignments). Failure to do so may result in an otherwise correct answer being rejected by the grader. In most cases where a seed is required, you will be reminded of this fact.

* + 1. Using the mean Function for Probability

In R, applying the mean function to a logical vector returns the proportion of elements that are TRUE. It is very common to use the mean function in this way to calculate probabilities and we will do so throughout the course.

Suppose you have the vector beads from a previous video:

*beads <- rep(c("red", "blue"), times = c(2,3))*

*beads*

*[1] "red" "red" "blue" "blue" "blue"*

To find the probability of drawing a blue bead at random, you can run:

*mean(beads == "blue")*

*[1] 0.6*

This code is broken down into steps inside R. First, R evaluates the logical statement beads == "blue", which generates the vector:

*FALSE FALSE TRUE TRUE TRUE*

When the mean function is applied, R coerces the logical values to numeric values, changing TRUE to 1 and FALSE to 0:

0 0 1 1 1

The mean of the zeros and ones thus gives the proportion of TRUE values. As we have learned and will continue to see, probabilities are directly related to the proportion of events that satisfy a requirement.

* + 1. Probability Distributions

Distribution for categorical outcomes is relatively straight forward. We can assign a probability to each category. In cases that can be thought of as beads in an urn, for each bead type, the proportions defines the distribution these proportions define the probability for each group. However, later in applications that are more common in data science, we will learn about probability distribution for continuous variables.

* + 1. Independence

We say that 2 events are independent if the outcome of one does not affect the other. The classic examples are coin tosses. Every time we toss a fair coin, the probability of seeing heads is ½ regardless of what previous tosses have revealed. The same is true when we pick beads from an urn with replacement. In the example we saw that the probability of red is 0.4 regardless of previous draws. Many of examples of events that are not independent come from card games. When we deal the first card, the probability of getting, say, a king is 1 in 13. This is because there are 13 possibilities. You can get an ace, a 2, a 3, a 4,…,10, jack, queen or king. Now if we deal a king for the first card and I don`t replace it, then the problem of getting a king in the 2nd card is less because there are only 3 kings left. The probability is 3 out of 51. These events are, therefore, dependent. When events are dependent, conditional probabilities are useful and necessary to make correct calculations. We already saw an example of a conditional probability. We computed the probability that a second dealt card is a king given that the first was a king. We use the following notation:

Note that when we have independent event, let’s say A and B, the probability of A given B is:

B has no influence of the outcome of A. If we want to know the probability of two event occurring A and B, the probability of A and B is:

Let`s use blackjack as an example, In blackjack you get 2 random cards without replacement, then you can ask for more, the goal is to get closer to 21 that the dealer without getting over it. Face cards and 10 are worth 10 points and Ace are worth either 1 or 11.So if you get an ace and a face card, you win automatically. To calculate the chance of having a blackjack, we compute the probability of getting an ace has the first card and then multiply by the probability of getting a face card or a 10 given that the 1st was an ace. Which is:

This multiplicative rule applies to even more than 2 events for example:

When we have independent events, the multiplicative rule becomes simpler:

Becareful when using the multiplicative rule in practice. When we are assuming independence and there is none the results can be very different and be incorrect. For example Imagine a court case in which the suspect is described as having a moustache and a beard, and the prosecution brings an expert that because 1 in 10 men have beards and 1 in 5 men have moustache, using the multiplicative rule, only 2% of men have both beards and moustaches. However to multiply this way means that we assume independence of events, but that’s not the case. The conditional probability of a man having a mustache conditional on them having a beard is quite high. It’s about 95%. So the correct calculation is 0.2\*0.95≈0.09

* 1. Combinations and Permutations
     1. Combinations and Permutations

So far, we imagine an urn with 5 beads, 3 blue 2 red. To compute the probability distribution of 1 draw, we simply listed out all the possibilities, there were 5. For each event, we counted how many of these possibilities were associated with that event. So for example, for the blue beads, the probability is 0.6. For more complicated examples, however, these computations are not necessarily straightforward. For example, what does the probability that if I draw 5 cards without replacement, I get all cards of the same suit, this is called a flush in poker ?

Discrete probability teaches us how to make these computations using math. Here we focus on how to use R code. Let`s first define a deck of card. To do this we use the functions **expand.grid()** and **paste()**. Paste() creates strings by joining smaller strings, for example:

*number<-"Three"*

*suit<-"Hearts"*

*paste(number,suit)*

*[1] "Three Hearts"*

The function expand.grid() give us all the combinations of 2 lists, for example:

*expand.grid(pants=c("blue","black"),shirt=c("white","grey","plaid"))*

*pants shirt*

*1 blue white*

*2 black white*

*3 blue grey*

*4 black grey*

*5 blue plaid*

*6 black plaid*

So to generate a deck of card we do :

*suits<-c("Diamonds","Clubs","Hearts","Spades")*

*number<-c("Ace","Deuce","Three","Four","Five","Six","Seven","Eight","Nine","Ten","Jack","Queen","King")*

*deck<-expand.grid(number=number,suits=suits)*

*deck<-paste(deck$number,deck$suits)*

Now that we have a deck, let’s start by doing something simple. Let’s double check that the probability of a king in the first is 1 in 13. We simply compute the proportion of possible outcomes that satisfy our condition. So we create a vector that contains the four ways we can get a king and then simply check the proportion of the deck is one of these cards:

*kings<-paste("King",suits)*

*mean(deck %in% kings)*

*[1] 0.07692308*

Now, how about the conditional probability of the second card being a king, given that the first was a king? Earlier we deduced that if 1 king is already out, then there’s 51 left. So, the probability is 3 in 51. But let’s confirm by listing out all possible outcomes. Let’s confirm it. To do this we are going to use the **combinations() and permutations()** functions that are available from the gtools package. The permutations function computes for any list of size n all the different way we can select R items. For example:

*library(gtools)*

*> permutations(5,2)*

*[,1] [,2]*

*[1,] 1 2*

*[2,] 1 3*

*[3,] 1 4*

*[4,] 1 5*

*[5,] 2 1*

*[6,] 2 3*

*[7,] 2 4*

*[8,] 2 5*

*[9,] 3 1*

*[10,] 3 2*

*[11,] 3 4*

*[12,] 3 5*

*[13,] 4 1*

*[14,] 4 2*

*[15,] 4 3*

*[16,] 4 5*

*[17,] 5 1*

*[18,] 5 2*

*[19,] 5 3*

*[20,] 5 4*

Notice that the order matters, 3,1 is different than 1,3. Also 1,1; 2,2; 3,3 are not there because once we pick a number, it can’t appear again

The function can take a vector, so for example if you want to see 5 random 7-digit phone numbers out of all possible phone numbers here is how to do it:

all\_phone\_numbers<-permutations(10,7,v=0:9)

n<-nrow(all\_phone\_numbers)

index<-sample(n,5)

all\_phone\_numbers[index,]

[,1] [,2] [,3] [,4] [,5] [,6] [,7]

[1,] 3 2 7 6 8 4 5

[2,] 1 8 0 5 2 4 6

[3,] 1 0 7 4 2 5 6

[4,] 4 7 8 9 1 2 3

[5,] 2 7 3 5 0 8 4

To compute all possible ways that we can choose 2 cards when the order matters, we simply type:

*hands<-permutations(52,2,v=deck)*

Now, we’re going to define the first card and the second card by grabbing the first and second columns:

*first\_card<-hands[,1]*

*second\_card<-hands[,2]*

And now we can, for example, check how many cases have a first card that is a king:

*sum(first\_card %in% kings)*

*[1] 204*

And to find the conditional probability, we ask what fraction of these 204 have also a king in the second card:

*sum(first\_card %in% kings & second\_card %in% kings)/sum(first\_card %in% kings)*

*[1] 0.05882353*

This is an R version of the multiplication rule, which tell us:

Now, what if the order does not matter? For example, in blackjack, if you get an ace and a face card or a 10, it’s called a natural 21 and you win automatically. If we want to compute of this happening, we want to enumerate the combinations not permutations, since the order doesn’t matter.

So we use combinations notice the difference:

*permutations(3,2)*

*[,1] [,2]*

*[1,] 1 2*

*[2,] 1 3*

*[3,] 2 1*

*[4,] 2 3*

*[5,] 3 1*

*[6,] 3 2*

*> combinations(3,2)*

*[,1] [,2]*

*[1,] 1 2*

*[2,] 1 3*

*[3,] 2 3*

So to compute the probability of a natural 21 in blackjack, we can do this:

*aces<-paste("Ace",suits)*

*> facecard<-c("King","Queen","Jack","Ten")*

*> facecard<-expand.grid(number=facecard,suit=suits)*

*> facecard<-paste(facecard$number,facecard$suit)*

*> hands<-combinations(52,2,v=deck)*

*mean(hands[,1]%in%aces & hands[,2]%in% facecard)*

*[1] 0.04826546*

Instead of using combinations to deduce the exact probability of natural 21, we can also use a Monte Carlo to estimate this probability. In this case, we draw two cards over and over and keep track of how many 21’s we get. We can use the function sample to draw a card without replacement like this:

*hand<-sample(deck,2)*

*> hand*

*[1] "Ace Hearts" "Nine Clubs"*

We didn’t get a 21 there. We can repeat this over and over and we get a very good approximation

*results<-replicate(B,{*

*+ hand<-sample(deck,2)*

*+ (hand[1]%in% aces & hand[2]%in% facecard)|*

*+ (hand[2] %in% aces & hand[1]%in%facecard)*

*+ })*

*> mean(results)*

*[1] 0.05*

* + 1. The Birthday Problem

Suppose you’re in a classroom with 50 people. If we assume this a randomly selected group, what is the chance that at least 2 people have the same birthday?

Although it is somewhat advanced, we can actually deduce this mathematically, and we do this later, but now we are going to use Monte Carlo simulations. For simplicity, we assumed that nobody was born on February 29th. The birthday can be represented as numbers between 1 and 365. So a sample of 50 random birthdays can be obtained simply using the sample function:

*n<-50*

*bdays<-sample(1:365,n,replace = TRUE)*

To check if, in this set of 50 people we have at least two with the same birthday we can use the function **duplicated()**, which returns true whenever an element of a vector has already appeared in that vector. For example:

*duplicated(c(1,2,3,1,4,3,5))*

*[1] FALSE FALSE FALSE TRUE FALSE TRUE*

*[7] FALSE*

So to check if two birthdays were the same, we simply use **any()** and **duplicated()** like this:

*any(duplicated(bdays))*

*[1] TRUE*

We did have at least 2 people, with the same birthday. Now, to estimate the probability, we`re going to repeat the experiment. So what we do is we do it 10000 times, we use the replicate function like this:

*results<-replicate(B,{*

*+ bdays<-sample(1:365,n,replace = TRUE)*

*+ any(duplicated(bdays))*

*+ })*

*> mean(results)*

*[1] 0.9698*

* + 1. Sapply

Let’s say that you want to bet with a friends about 2 people having the same birthday in a group of people. When are the chances largen than 50%? Larger than 75%? Let’s create a lookup table. We can quickly create a function to compute this for any group. We write a function like this:

*> compute\_prob<-function(n,B=10000){*

*+ same\_day<-replicate(B,{*

*+ bdays<-sample(1:365,n,replace = TRUE)*

*+ any(duplicated(bdays))*

*+ })*

*+ mean(same\_day)*

*+ }*

Now, we want to compute this function, we want to apply this function to several values of n from 1 to 60, let’s define n:

*n<-seq(1:60)*

Now, we can use a for loop to apply this function to each value of n. In R loops are rarely the preferred approach, we try to perform operation on entire vectors. Arithmetic operations, for example operate on vectors in an element wise fashion. Unfortunately, the function we just wrote does not work element wise, since it’s expecting an n. If we type compute prob and send it the vector n, we will not get what we want, we will get just one number:

*compute\_prob(n)*

*[1] 0*

What we can do instead is use the function **sapply()**, it permit us to perform element-wise operations on any functions. Here’s how it works. We’ll define a simple example for the vector 1 through 10:

*x<-1:10*

*> sapply(x,sqrt)*

*[1] 1.000000 1.414214 1.732051 2.000000*

*[5] 2.236068 2.449490 2.645751 2.828427*

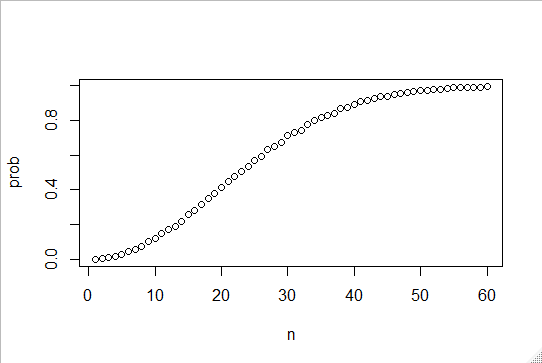
*[9] 3.000000 3.162278*

Of course, we don’t need to do that because we sqrt function works element wise, so for our case we can simply type:

*prob<-sapply(n,compute\_prob)*

*plot(n,prob)*

1. Probability of two people having the same birthday against the size of the group



Now let’s compute the exact probability rather than use Monte Carlo simulations. We can use what we’ve learned about probability theory to compute the exact value. To make the math simpler, we’ll compute the probability of it not happening instead of happening.

If we continue this way and find the chances of all, say 50 people, having unique birthdays :

Now, we can easily write a function that does this. This time we call it exact prob:

*exact\_prob<-function(n){*

*+ prob\_unique<-seq(365,365-n+1)/365*

*+ 1-prod(prod\_unique)*

*+ }*

Now we can compute each probability for each n using sapply again, like this:

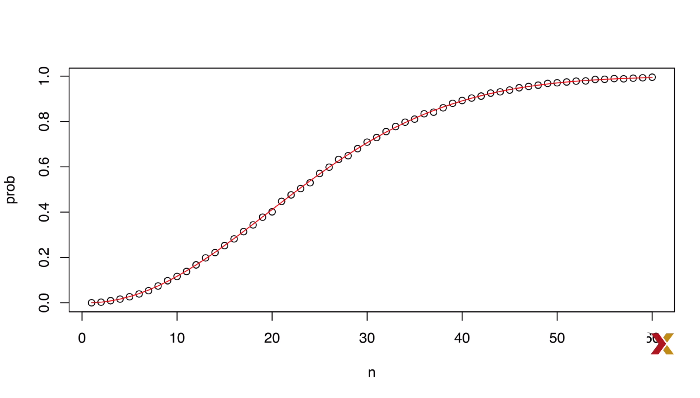
*eprob<-sapply(n,exact\_prob)*

And if we plot it, we can see that the Monte Carlo simulation were almost exactly right:

*plot(n,prob)*

*lines(n,eprob,col="red")*

1. Comparisons of Monte Carlo simulation and Mathematical calculation of probability of having 2 people with the same birthday



* + 1. How Many Monte Carlo Experiments are Enough?

So far, we have used 10000 Monte Carlo experiments. It turns out that this provide an accurate estimate for the examples we looked at. In more complex calculations, 10000 may not nearly be enough. Also, for some calculations, 10000 experiments might not be computationally feasible, and it might me bore than what we need. In practice we won’t know what the answer is, so we don’t know if our Monte Carlo estimate is accurate. We know that the larger the number of experiments, we’ve been using the letter B to represent that, the better the approximation. But how big do we need it? Answering that question often requires advanced theoretical statistics training. One practical approach will describe is to check for the stability of the estimate. Here’s an example, we are going to estimate the probability of 2 people having a certain birthday using different sizes of the Monte Carlo simulations:

*B<-10^seq(1,5,len=100)*

*compute\_prob<-function(B,n=22){*

*+ same\_day<-replicate(B,{*

*+ bdays<-sample(1:365,n,replace=TRUE)*

*+ any(duplicated(bdays))*

*+ })*

*+ mean(same\_day)*

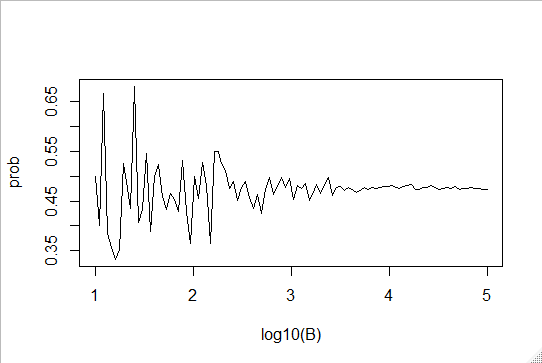
*+ }*

We compute the simulation, and now look at the values that we get for each simulation:

prob<-sapply(B,compute\_prob)

> plot(log10(B),prob,type="l")

1. Estimation of the correct number of Monte Carlo simulations



We can see on the graph that the line stabilize around x=4 (so log10(4)=10000)

* 1. Addition Rule and Monty Hall
     1. The Addition Rule

Earlier, we showed how to compute the probability of a natural 21 in blackjack. Here we’re going to show you the addition rule, this rule tell us that:

We will need that because you can get 21 in 2 ways either a face card and then ace or an ace and a facecard. Let’s apply the formula to the natural 21 calculation. Since both hands can’t happen at the same time (you don’t have 4 cards) the calculation becomes:

And

So

The result is the same for both, we get the same result that we got before for a natural 21 which is ≈0.05

* + 1. The Monty Hall Problem

In the 70s there was a game show called let’s make a deal. Monty Hall was the host- this were the name of the problem comes from. At some point in the game, contestants were asked to pick one of three doors. Behind one door. There was a prize. The other two had a goat behind them. If the contestants did not pick the prize door on his or her first try, Monty Hall would open one of the two remaining doors and show the contestant that there was no prize behind that door. So you’re left with 2 doors, the one you picked and one door that you do not know what’s behind it. So, then Monty Hall would ask, do you want to switch doors?

We can use probability to show that if you stick to the original door, your chances of winning a car or whatever big prize is 1 in 3 but if you switch you chances double to 2 in 3. Here we are going to use a Monte Carlo simulation to show you that this is the case.

So let’s create a simulation that imitates the strategy of sticking to the same door:

stick<-replicate(B,{

+ doors<-as.character(1:3)

+ prize<-sample(c("car","goat","goat"))

+ prize\_door<-doors[prize=="car"]

+ my\_pick<-sample(doors,1)

+ show<-sample(doors[!doors %in% c(my\_pick,prize\_door)],1)

+ stick<-my\_pick

+ stick==prize\_door

+ })

*mean(stick)*

*[1] 0.3361*

Now let’s repeat the exercise but with the switch strategy:

*switch<-replicate(B,{*

*+ doors<-as.character(1:3)*

*+ prize<-sample(c("car","goat","goat"))*

*+ prize\_door<-doors[prize=="car"]*

*+ my\_pick<-sample(doors,1)*

*+ show<-sample(doors[!doors %in% c(my\_pick,prize\_door)],1)*

*+ stick<-my\_pick*

*+ switch<-doors[!doors%in%c(my\_pick,show)]*

*+ switch==prize\_door*

*+ })*

*> mean(switch)*

*[1] 0.6631*

1. Section 2: Continuous Probability
   1. Continuous Probability
      1. Continuous Probability

When summarizing a list of numeric values such as heights, it’s not useful to construct a distribution that assigns a proportion to each possible outcome. Note, for example, that if we measure every single person in a very large population with extremely high precision, because no 2 people are exactly the same height, we would need to assign a proportion to each observed value. Similarly when defining probability distributions, it is not useful to assign a very small probability to every single height. It is much more practical to define a function that operates on intervals rather than single values. The standard way of doing this is using the **cumulative distribution function** (CDF). We previously described the empirical cumulative distribution function (eCDF) as a basic summary of a list of numeric values. As an example:

*data("heights")*

*x<-heights%>%filter(sex=="Male")%>%.$height*

Now we can define the ECDF like this:

f*<-function(a) mean(x<=a)*

Note that we have not yet introduced probability, we’ve been talking about list of numbers and summarizing these lists. So let’s introduce probability. For example let’s ask if I pick one of the male students at random, what is the chance that he is taller than 70.5 inches?

Using eCDF we have:

*1-f(70)*

*[1] 0.377*

Once the eCDF is compute we can use it to compute the probability of any subset. For example the probability of a student being between the height a and the height b is simply:

*f(b)-f(a)*

* + 1. Theoretical Distribution

In the data visualization module we introduced the normal distribution as a useful approximation to many naturally occurring distributions, including heights. The cumulative distribution for the normal distribution is defined by a mathematical formula:

Where is the standard deviation and µ the average. In R it can be obtained by the **pnorm()** function. We say that a random quantity is normally distributed with average, avg and standard deviation, s, if its probability distribution is defined by:

*F(a)=pnorm(a,avg,s)*

This is useful because if we are willing to use the normal approximation for say, height, we don’t need the entire dataset to answer questions such as, what is the probability that a randomly selected student is taller than 70.5 inches. We just need the average height and the standard deviation like this:

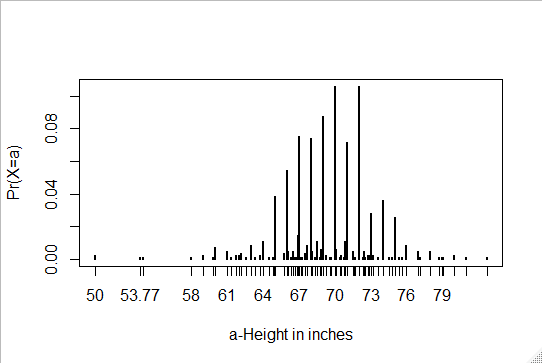
*1-pnorm(70.5,mean(x),sd(x))*

*[1] 0.371*

The normal distribution is defined for continuous variables. It is not described for discrete variables. However, for practicing data scientists, pretty much everything we do involves data, which is technically speaking discrete. For example we could consider our data categorical with each specific height a unique category. The probability distribution would then be defined by the proportions of students reporting each of those unique heights. We do it like this:

*plot(prop.table(table(x)),xlab="a-Height in inches",ylab="Pr(X=a)")*

1. proportions of students reporting unique heights



Most student rounded up their heights to the nearest inch, others reported values with much more precision. For example, a student reported his height to be 69.6850393700787. Which is very precise, well it turns out that’s 177 cm. So he converted his measure in cm to inches and copied pasted the result into the place where they had to reports their heights. The probability assigned to this height is about 0.001. It’s 1 in 708. However, the probability for 70 inches is 0.12. This is much higher than what was reported with this other value. But does it really make sense to think that the probability of being exactly 70 inches is so much higher that the probability of being 69.68? Clearly, it is much more useful for data analytics purposes to treat this outcome as a continuous numeric variable. With continuous distributions, the probability of a singular value is not even defined. Instead, we define probabilities for intervals.

So we could ask instead, what is the probability that someone is between 69.99 and 70.001. In cases like height in which the data is rounded, the normal approximation is particularly useful if we deal with intervals that include exactly 1 round number , for example:

*mean(x<=68.5)-mean(x<=67.5)*

*[1] 0.114532*

*> mean(x<=69.5)-mean(x<=68.5)*

*[1] 0.1194581*

*> mean(x<=70.5)-mean(x<=69.5)*

*[1] 0.1219212*

When we compare it to the normal approximation:

*pnorm(68.5,mean(x),sd(x))-pnorm(67.5,mean(x),sd(x))*

*[1] 0.1031077*

*> pnorm(69.5,mean(x),sd(x))-pnorm(68.5,mean(x),sd(x))*

*[1] 0.1097121*

*> pnorm(70.5,mean(x),sd(x))-pnorm(69.5,mean(x),sd(x))*

*[1] 0.1081743*

We get almost the same result. The approximation, however, is not that useful for other intervals. For example, those that don’t include an integer, here are 2 examples:

*mean(x<=70.9)-mean(x<=70.1)*

*[1] 0.02216749*

*pnorm(70.9,mean(x),sd(x))-pnorm(70.1,mean(x),sd(x))*

*[1] 0.08359562*

We call this situation discretization. Although the true height distribution is continuous, the reported heights tend to be more common at discrete values, in this case, due to rounding. If we are aware of how to deal with this reality. The normal approximation can still be a very useful tool.

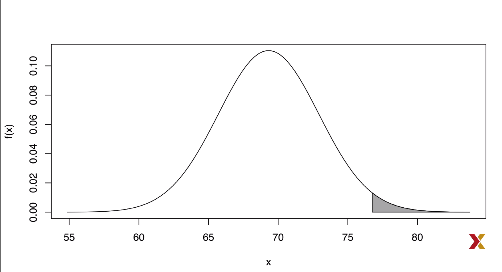
* + 1. Probability Density

For categorical data, we can define the probability of a category. For example, a roll of a die, let’s call it x, can be 1,2,3,4,5 or 6. The probability of 4 is defined as probability of x equals 4 is 1/6. The CDF can be easily be defined by simply adding up probability. So:

In contrast, for continuous distributions the probability of a single value is not defined. However, there is a theoretical definition that has a similar interpretation. This has to do with the probability density. The probability density at x is defined as follow:

For example to estimate the probability of someone being taller than 76 inches, we can use the probability density. Mathematically, the gray are in the figure below represent the probability of x being bigger than 76.

1. Probability density function of height with probability of x being bigger than 76 inches represented



In R to get the probability density function for the normal distribution you use the **dnorm()** function

* + 1. Monte Carlo Simulations

Here we are going to show how to run Monte Carlo simulations using normally distributed variables. R provides a function to generate normally distributed outcomes with the function **rnorm(size, average=0,standard\_deviation=1)** here is an example:

*x<-heights%>%filter(sex=="Male")%>%.$height*

*> n<-length(x)*

*> avg<-mean(x)*

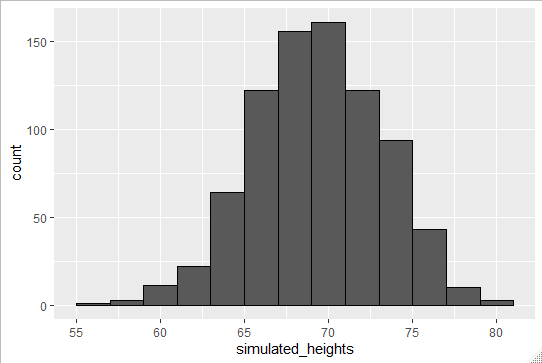
*> s<-sd(x)*

*> simulated\_heights<-rnorm(n,avg,s)*

Not surprisingly the distribution of these outcomes looks normal:

*data.frame(simulated\_heights=simulated\_heights)%>%ggplot(aes(simulated\_heights))+geom\_histogram(color="black",binwidth = 2)*

1. Distribution of heights generated by rnorm



This is one of the most useful functions in R, as it will permit us to generate data that mimics naturally occurring events, and it’ll let us answer questions related to what could happen by chance by running Monte Carlo simulations. For example, if we pick 800 males at random, what is the distribution of the tallest person? Specifically, we could ask, how rare is that the tallest person is a seven-footer? We can use the following Monte Carlo simulation to answer this question:

We’re going to run 10000 simulations

*B<-10000*

and for each one, we’re going to generate 800 normaly distributed values

*tallest<-replicate(B,{*

*+ simulated\_data<-rnorm(800,avg,s)*

pick the tallest one and return that

*max(simulated\_data)*

*+ })*

Now we can ask, what proportion of these simulations return a seven-footer as the tallest person?

*mean(tallest>=7\*12)*

*[1] 0.0185*

* + 1. Other Continuous Distributions

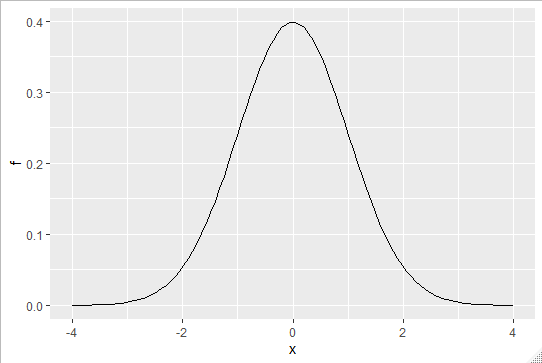
The normal distribution is one of many theoretical distributions which includes: **student-t, chi-squared, exponential, gamma, and beta distribution**. R provides function to compute the **d**ensity, the **q**uantiles, the cumulative **p**robability distribution function, and generate Monte Carlo simulations for all these distributions. R use a conventions that let us remember the names of these functions. Namely using the letters **d, q, p**  and **r** for density, quantile, probability density function and random, by putting these letters in front of a shorthand for the distribution, five us the names of these useful functions.

We’ve already seen the function dnorm, pnorm and rnorm, which give us density, probability and random for the normal distribution, the function **qnorm()** give us the quantiles. For example, we can use the dnorm function to generate this plot:

*x<-seq(-4,4,length.out = 100)*

*> data.frame(x,f=dnorm(x))%>%ggplot(aes(x,f))+geom\_line()*

1. Density function for the normal distribution



For the student’s t distribution, which has shorthand t, we can use the functions **dt, qt, pt and rt**

1. Section 3: Random Variables, Sampling Models, and the Central Limit Theorem
   1. Random Variables and Sampling Models
      1. Random Variables

Random variables are numeric outcomes from a random process. We can easily generate random variable using some the simple examples we have shown, such as the red and blue bead urn. For example, define x<-1 if a bead is blue, and red otherwise. Here is the code:

*beads<-rep(c("red","blue"),times=c(2,3))*

*> X<-ifelse(sample(beads,1)=="blue",1,0)*

X is a random variable. Every time we select a new bead, the outcome changes randomly. Sometimes it’s 1 sometimes it’s 0. Here’s some examples: ifelse(sample(beads,1)=="blue",1,0)

*[1] 0*

*> ifelse(sample(beads,1)=="blue",1,0)*

*[1] 1*

*> ifelse(sample(beads,1)=="blue",1,0)*

*[1] 1*

In data science, we often deal with data that is affected by chance in some way. The data comes from a random sample, the data is affected by measurement error, or the data measures some outcome that is random in nature. Being able to quantify the uncertainty introduced by randomness is one of the most important jobs of a data scientist. Statistical inference offers a framework for doing this, as well as several practical tools. First, we need to learn how to mathematically describe random variables.

We start with games of chances as an illustrative example.

* + 1. Sampling Models

Many data-generation procedures can be modeled quite well as draws from an urn. For example, we can model the process polling likely voters as drawing 0’s, Republican, 1’s Democrats, from an urn containing 0 and 1 code for all likely voters.

In epidemiological studies, we often assume that the subjects in our study are a random sample from the population of interest. The data related to a specific outcome can be modeled as a random sample from urn containing the values for those outcomes for the entire population. Similarly, in experimental research we assume that the individual organism we are studying are a random sample form a larger population. Randomized experiments can also be modeled by draws from urn, given the way individuals are assigned to groups. When getting assigned you draw your group at random. Sampling models are therefore ubiquitous in data science.

Let’s suppose a very small casino hires you to consult on whether they should set up a roulette wheel. They want to know if they can make money off it. Or if it’s too risky. We will assume that 1000 people will play and that the only game you can play is to bet on red or black. They want to know how much money they will make or lose.

We are going to define a random variable, S, that will represent the casino’s winnings. Let’s start by constructing the urn for our sampling model. A roulette wheel has 18 red pockets, 18 black pockets and 2 green ones. So playing a color in one game of roulette is equivalent to drawing the urn:

*color<-rep(c(“Black”,”Red”,”Green”),c(18,18,2))*

The 1000 outcomes from 1000 people playing are independent draws from this urn. If red comes up the gambler wins and the casino loses 1$ so we draw a -1. Otherwise the casino wins $1 and we draw a 1:

*n<-1000*

*X<-sample(ifelse(color=="Red",-1,1),n,replace=TRUE)*

Here are the first 10 outcomes of these 1000 draws:

*X[1:10]*

*[1] 1 1 -1 -1 -1 1 1 -1 1 1*

Because we know the proportions of 1’s and -1’s, inside the urn, we can generate the draws with one line of code without defining color:

*X<-sample(c(-1,1),n,replace=TRUE,prob=c(9/19,10/19))*

We call this approach a **sampling model** since we are modelling the random behavior of a roulette with the sampling of draws form an urn. The total winnings, S, is simply the sum of these 1000 independent draws, so here is an example of S:

*S<-sum(X)*

*> S*

*[1] 74*

Note that if you run that code again, you’ll get a different value of S. This is because, of course, S is a random variable.

A very useful concept is the **probability distribution of the random variable**. It tell us the probability of the observed value falling in any given interval. For example, if we want to know the probability that we lose money, we’re asking, what is the probability that S is in the interval S <0

If we define a CDF let’s call it:

We’ll be able to answer any question related to the probability of events defined by a random variable S, including the event S<0. We call f the random variable’s distribution function. We can estimate the distribution function for the random variable S by using a Monte Carlo simulation to generate a lot of the random variable like this:

*n<-1000*

*> B<-10000*

*> S<-replicate(B,{*

*+ X<-sample(c(-1,1),n,replace=TRUE,prob=c(9/19,10/19))*

*+ sum(X)*

*+ })*

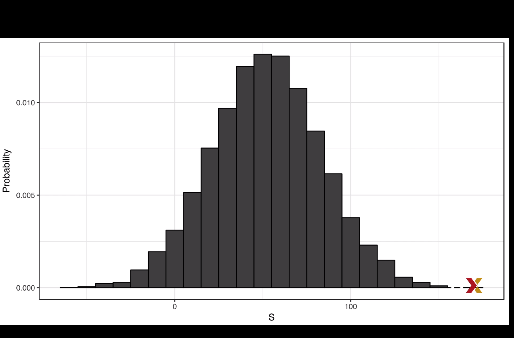
Now we can ask in our simulation how often did we get sums smaller or equal to a? Like this:

*mean(S<=0)*

*[1] 0.0506*

In fact, we can visualize the distribution by creating a histogram showing the probability f(b)-f(a) for several interval ab. Here it is:

1. Histogram of Probability of F(b)-F(a) for several interval ab



Now we can answer the casino’s question, how likely is it that they lose money? They answer:

*mean(S<=0)*

*[1] 0.0506*

From the histogram, we can also see that distribution appears to be approximately normal. If you make a QQplot you’ll confirm that the normal approximation is close to perfect. If, in fact, the distribution is normal, then all we need to define is the distribution’s average and standard deviation. We can easily compute these from the original values from which the distribution is created:

*mean(S)*

*[1] 52.7478*

*> sd(S)*

*[1] 31.51562*

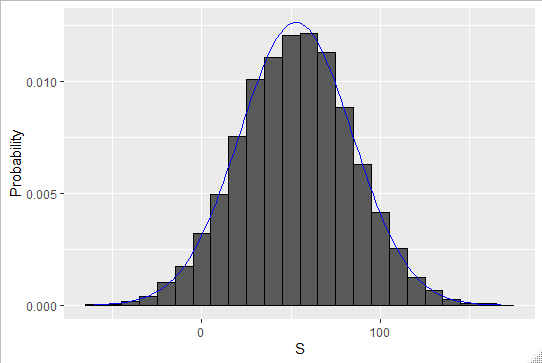
If we add normal density with this average and standard deviation to the histogram we created earlier, we see that it matches very well:

*s<-seq(min(S),max(S),length=100)*

*normal\_density<-data.frame(S=S,f=dnorms(S,mean(S),sd(S)))*

*data.frame(S=S)%>%ggplot(aes(S,..density..))+geom\_histogram(color="black",binwidth = 10)+ylab("Probability")+geom\_line(data=normal\_density,mapping = aes(S,f),color="blue")*

1. Histogram of Probability of F(b)-F(a) for several interval ab with density function



The average and the standard

* + 1. Distributions versus Probability Distributions

We need to make a distinction between the distribution of a list of numbers, which we covered in the module Visualization, and a probability distribution. Previously we described how any list of numbers, let`s say x1,…,xn has a distribution. The definition is quite straightforward. We define F(a) as a function that answers the question, what proportion of the list is less than or equal to a? When the distribution is approximately normal, we define the average and the standard deviation. These are defined with a straightforward operation of the list.

In R we simply compute the average and standard deviation like this for example:

*avg<-sum(x)/length(x)*

*s<-sqrt(sum((x-avg)^2)/length(x))*

A random variable x has a distribution function. To define this, we do not need a list of numbers. In this case, to define the distribution, we define F(a) as a function that answers the question what is the probability that x is less than or equal to a? There is no list of numbers. However, if x is defined by drawing from an urn with numbers in it, then there is a list, the list of numbers inside the urn. The distribution of that list, in this case, is the probability distribution of x and the average and standard deviation of that list are the expected value and standard errors of the random variable.

Another way to think about it, that does not involve an urn, is to run a Monte Carlo simulation and generate a large list of outcomes of x. These outcomes are a list of numbers. The distribution of this list will be a very good approximation of the probability of x. The longer the list the better the approximation. The average and the standard deviation of this list will approximate the expected value and standard error of the random variable.

* + 1. Notation for Random Variables

In statistics, capital letters are used to denote random variables and lower case letters are used for observed values. You’ll see some notation that include both. For example you’ll see events defined as:

Here X is a random variable, making it a random event, and x is an arbitrary value and not random. So for example, X might represent the number on a die roll , that’s a random value, and x will represent an actual value we see. So in this case:

* + 1. Central Limit Theorem

**The Central Limit Theorem or (CLT)** tell us that when the number of independent draws also called sample size is large, the probability distribution of the sum of these draws is approximately normal. Because sampling models are used for so many data generation processes, the CLT is considered one of the most important mathematical insights in history.

Previously, we discussed that if we know the that the distribution of a list of numbers is approximated by the normal distribution, all we need to describe the list are the average and standard deviation, we’ve seen that the same applies to probability distributions. We have described sampling models for draws. We will now go over the mathematical theory that let us approximate the probability distribution for the sum of draws. Once we do this, we will be able to help the casino predict how much money they will make.

The same approach we use for sum of draws will be useful for describing the distribution of averages and proportions, which we will need to understand for example, how polls work.

The first important concept to learn is the **expected value**, in statistics, it is common to use:

Where is the expected value of the random variable. A random variable will vary around an expected value in a way that if you take the average of lots and lots of draws the average of the draws will approximate the expected value. A useful formula is that the expected value of the random variables defined by one draw is the average of the numbers in the urn. For example, in our urn used to model betting on red on roulette, we have 20 1’s and 18 -1’s so the expected value is:

It is a bit counterintuitive to say that X varies around 0.05 when the only values it takes is 1 and -1. An intuitive way to think about the expected value is that if we play the game over and over, the Casino wins, on average, $0.05 per game. Our Monte Carlo Simulation confirms this:

*B<-10^6*

*> X<-sample(c(-1,1),B,replace = TRUE, prob=c(9/19,10/19))*

*> mean(X)*

*[1] 0.053102*

In general, if the urn has just 2 possible outcomes, a and b, with the proportions p and 1-p respectively, then the average is:

To see this, notice that if there are nb’s in the urn, then we have npas’s and n times -1pb’s, and because the average is the sum, we have:

Which give us the formula above if we simply it by n. The reason we define the expected value is because this mathematical definition turns out to be useful for approximating the probability distributions of sums, which in turns is useful to describe the distribution of averages and proportions. The first useful fact is that the expected value of the sums of draws is:

So, if 1000 people play roulette, the casino expects to win, on average 1000\*$0.05=$50. But this is an expected value. How different can one observation be from the expected value? The casino need to know this. What is the range of possibilities? If the negative numbers are too likely, we may not install the roulette wheels. Statistical theory, once again, answers this question. The **standard** **error (or SE)** give us an idea of the size of the variation around the expected value. If our draw are independent then the standard error of the sum is given by:

If an urn contains 2 values, a and b, with proportions p and 1-p respectively, the standard deviation is:

So, in our roulette example, the standard deviation is:

The standard error tell us the typical difference between a random variable and its expectation. So because 1 draw is obviously the sum of 1 draw, we can use a formula to calculate that the random variable by 1 draw has an expected value of $0.05 and a standard error of about 1. This make sense since we either get a 1 or -1 with slightly favored over the minus 1. Using the formula, the sum of 1000 people playing has standard error of about 32$. So when 1000 people bet on red, the Casino is expected to win $50 with a standard error of 32$. We still can’t answer the question how likely is the Casino to lose money? Here the CLT help. It tell us that the distribution of the sum of S is approximated by normal distribution. Using the formula, we know that the expected value and standard errors are $52 and $32 respectively. Note that the theoretical values match those obtained with Monte Carlo simulation ran earlier. With the CLT we can compute the probability of the Casino losing money using the approximation. We write:

*n<-1000*

*> mu<-n\*(20-18)/38*

*> se<-sqrt(n)\*2\*sqrt(90)/19*

*> pnorm(0,mu,se)*

*[1] 0.04779035*

About 5%.

* 1. The Central Limit Theorem Continued
     1. Averages and Proportions

There are some useful mathematical results from the CLT that we are going to list:

Property 1: The expected value of the sum of random variables is the sum of the expected values of the individuals random variables

We can write like this:

If the x’s are drawn from the same urn, then they all have the same expected value. We call it mu here, and therefore:

Property 2: The expected value of a random variable times a non-random constant is the expected value times that non-random constant

We can write like this:

A consequence of the 2 property we described is that the expected value of the average of draws from the same urn is the expected value of the urn, call it mu again.

Property 3: The square of the standard error of the sum of independent random variables is the sum of the square of the standard error of each random variable

Or:

Note that:

Property 4: The standard error times a non-random constant is the standard error times a non-random constant.

Like this:

A consequence of the 2 property we described is that the standard error of the average of independent draws from the same urn is the standard deviation of the urn, call it sigma divided by the sqrt of n:

Property 5: if X is a normally distributed random variable, then if a and b are non-random constants, aX+b is also normally distributed

* + 1. Law of Large Numbers

One of the implication of the properties we just saw, is that the standard error of the average of draws becomes smaller as the number of draws n grows larger. When n is very large then the standard error is practically 0, and the average of the draws converges to the average of the urn. This is the law of large numbers, or law of averages. Note that this law is sometimes misinterpreted. For example. If you toss a coin 5 times and you see heads each time, you might hear someone argue that the next toss is probably a tail because of the law of averages. On averages, we should 50% heads and 50% tails so we need more tails to make up.

A similar argument would be to say that red is due on roulette after seeing black come up 5 times in a row. These events are independent, so the probability remain the same at each draw. The law of averages applies only when the number of draws is very large not in small samples. After a 10000000 toss you would see about 50% heads, regardless of what the first 5 were.

* + 1. How Large is Large in CLT?

The CLT works when the number of draws is large. But how large is large? 15,100, a million? In many cases as few as 30 draws is enough to make the CLT useful, in some other instances as few as 10 is enough. However, these should not be considered general rules. Note for example, that when the probability of success is very small, we need large sample size. Consider the example of lottery. In lottery the chance of winning is less than 1 in a million. Thousands of people play, so the number of draws is very large. So, the central limit should work. Yet the number of winners, the sum of the draws, range between 0 and in very extreme cases, 4. This sum is not well approximated by the normal distribution, so the CLT doesn’t apply with a very large sample size. This is generally true when the probability of success is very low.

1. Section 4: The Big Short
   1. The Big Short
      1. The Big Short: Interest Rates Explained

In a way the sampling models we’ve been talking about are also used by banks to decide interest rates. Let’s see how. Suppose you run a small bank that has a history of identifying potential homeowners that can be trusted to make payments. In fact, historically each year only 2% of your customer didn’t pay back the money you lent them (they defaulted). However, note that if you simply loan money to everybody without interest, you’ll end up losing money to this 2%. However by charging everybody just a bit extra you can make up for the losses incurred due to the 2% and also pay the employees at work to make these loans happen. You can also make profit, but if you set the interest rate too high your clients will go to another bank. We use al these facts and some probability theory to decide what interest rates we should charge. Suppose your bank will give out 1000 loans for 180000 this year. Also suppose that your bank loses, after adding up all the costs, $200000 per foreclosure. For simplicity, we assume that incudes all operational costs.

A sampling model for this scenario is coded like this:

*n<-1000*

*loss\_per\_foreclosure<--200000*

*p<-0.02*

*defaults<-sample(c(0,1),n,prob=c(1-p,p),replace = TRUE)*

*sum(defaults\*loss\_per\_foreclosure)*

*[1] -2800000*

We can easily construct a Monte Carlo simulation to get an idea of the distribution of this random variable:

*losses<-replicate(B,{*

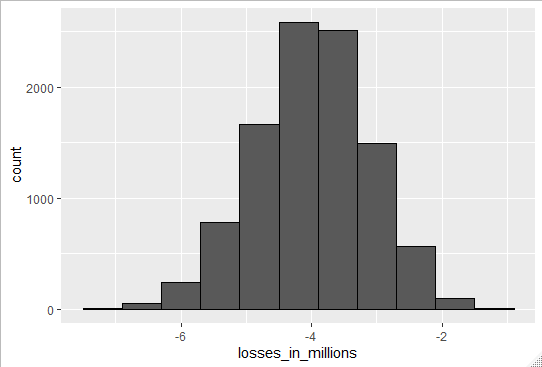
*+ defaults<-sample(c(0,1),n,prob=c(1-p,p),replace=TRUE)*

*+ sum(defaults\*loss\_per\_foreclosure)*

*+ })*

*data.frame(losses\_in\_millions=losses/10^6)%>%ggplot(aes(losses\_in\_millions))+geom\_histogram(binwidth=0.6,color="black")*

1. Distribution of Money lost in millions



Using what’ve learned about CLT we know that because our losses are a sum of independent draws, its distribution is approximately normal with expected value and standard deviation given by:

n\*(p\*loss\_per\_foreclosure+(1-p)\*0)

[1] -4e+06

> sqrt(n)\*abs(loss\_per\_foreclosure)\*sqrt(p\*(1-p))

[1] 885438

We can now set an interest rate to guarantee that on average we break even. Basically, we need to add a quantity x to each loan, which in this case are represented by draws, so that the expected value is 0. That mean breaking even. If we define l to be the loss per foreclosure, we need to set x so:

Which implies that x can be calculated by this R code:

*loss\_per\_foreclosure\*p/(1-p)*

*[1] -4081.633*

Which is interest rate of about 2%.

We still have a problem though. Although this interest rate guarantees that on average we break even, there’s a 50% change that we will lose money. So we need an interest rate that makes it unlikely for the bank to loose money. At the same time, if the interest rate is too high our client will go to another bank. So we must be willing to take some risks. So let’s say that we want our chances of losing money to be 1 in 100. What does x have to be now? We want:

We know that S is approximately normal. The expected value is given by that formula:

The standard error is given by that formula:

So we want:

We are going to substract the expected value of S and divide by the standard error on both sides of this equation:

The term on the left is now a standard normal random variable, which we will rename Z and fill in the blanks for the expected value and standard error:

l, p and n are all know values and Z is a normal random variable with expected value of 0 and standard error 1, it means that the quantity of the right must be equal to:

*qnorm(0.01,0,1)*

*[1] -2.326348*

If we set z to be qnorm of 0.01, it will give you a value of z for which the following formula is true:

So that means that the right side of the equation has to be equal to qnorm(0.01) so we have this formula:

Solving for x is now simple algebra. If we do it, we get that the x has to be about 6.249:

*l<-loss\_per\_foreclosure*

*z<-qnorm(0.01)*

*x<--l\*(n\*p-z\*sqrt(n\*p\*(1-p)))/(n\*(1-p)+z\*sqrt(n\*p\*(1-p)))*

*> x*

*[1] 6249.181*

Which is an interest rate of about 3%. Note also that by choosing this interest rate, we now have an expected profit of about $2 million:

*loss\_per\_foreclosure\*p+x\*(1-p)*

*[1] 2124.198*

We can run a Monte Carlo simulation and check our theoretical approximation. We do that and we indeed get that value again and again, the probability of profit being less than 0 according to Monte Carlo simulation is about 1%:

*B<-10000*

*profit<-replicate(B,{*

*+ draws<-sample(c(x,loss\_per\_foreclosure),n,prob = c(1-p,p),replace=TRUE)*

*+ sum(draws)*

*+ })*

*> mean(profit)*

*[1] 2122713*

*> mean(profit<0)*

*[1] 0.0146*

* + 1. The Big Short

One of our employees points our that since the bank is making about $2000 per loan, that you should give out more loans. Why just n?

You explained that finding those n clients was hard. You need a group that is predictable and that keep chances of defaults low. He then points out that even if the probability of default is higher, as long as your expected value is positive, you can minimize you chances of losing money by increasing n, the number of loans, relying on the law of large numbers. He claims that if the default rate is twice as high, say 4%, if we set the rate just a bit higher you will get a positive expected value. So if we set the interest rate at 5% we are guaranteed a positive value of $640 per loan. And we can minimize our chances of losing money. And we can minimize our chances of losing money by simply increasing the number of loans since:

And if we do the math, we will see that we can pick an n so that this probability is 1 in 100. Let’s see how:

We have:

And

So if z is qnorm(0.01) then we have the following formula:

Which tell us that if :

We are then guaranteed to have a probability of less than 0.01 of losing money. The implication is that as long as mu, the expected value is positive, we can find an n, a number of loans, that minimize the probability of loss. With x fixed now we can ask what n do we need for the probability to be 0.01? In our example we use the following formula

*z<-qnorm(0.01)*

*n<-ceiling((z^2\*(x-l)^2\*p\*(1-p))/(l\*p+x\*(1-p))^2)*

*n*

*[1] 22163*

This seems like a no-brainer. Your colleague decides to leave your bank, and start his own high-risk mortgage company. A few months later you colleague’s bank has gone bankrupt. A book is written about it and eventually a movie made relating to the mistake your friend and many others made. What happened? By making n large, we minimize the standard error of our per-loan profit. However, for this rule to hold, the X’s must be independent draws. The fact that one person defaults must be independent of other people defaulting.

To construct a more realistic simulation than the original one your friend ran, let’s assume there could be a global event that affects everybody with high-risk mortgages, and changes their probability. We will assume that with a 50-50 chance, all the probabilities go up or down slightly to somewhere between 0.03 and 0.05. But it happens to everybody at once, not just one person. These draws are no longer independent. Let’s use a Monte Carlo simulation to see what happens under this model:

*p<-0.04*

*> x<-0.05\*180000*

*> profit<-replicate(B,{*

*+ new\_p<-0.04+sample(seq(-0.01,0.01,length=100),1)*

*+ draws<-sample(c(x,loss\_per\_foreclosure),n,prob=c(1-new\_p,new\_p),replace=TRUE)*

*+ sum(draws)*

*+ })*

*mean(profit)*

*[1] 136171.9*

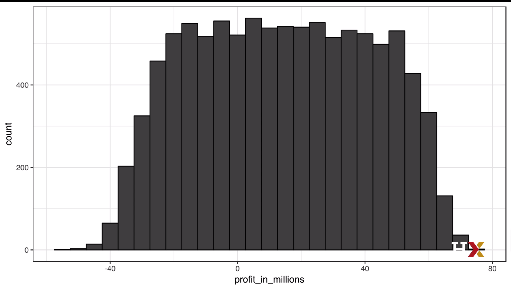
*mean(profit<0)*

*[1] 0.4062*

Note that we’re expected to make about 14 millions however the probability of the bank having negative earning shoots way up to 40%. Even scarier the probability of losing more than 10 millions is at 24% to understand how this happen,

we can look at the distribution of our random variable:

1. Distribution of random variable under high risk mortgages



The theory completely break downs, and our random variable has much more variability than expected.