Simulations in Empiricial Analysis

Dr. Aydede

Fall 2025 - SMU

In this lecture, we will learn how to simulate data and illustrate their use in several examples. More specifically we'll cover the following subjects:

1. **Sampling in R: `sample()`,**
2. **Random number generating with probablity distributions,**
3. **Simulation for statistical inference,**
4. **Creating data with a DGM,**
5. **Bootstrapping,**
6. **Power of simulation - A fun example.**

Why would we want to simulate data? Why not just use real data?

Because with real data, we don't know what the right answer is. Suppose we use real data and we apply a method to extract information, how do we know that we applied the method right?

Now suppose we create artificial data by simulating a **Data Generating Model** (DGM). Since we know the right answer (DGM), we can check whether or not our methods is right to extract the information we wish to have.

1. Sampling in R: `sample()`

Let's play with sample() for simple random sampling. We will see the arguments of `sample()` function.

```
sample(c("H","T"), size = 8, replace = TRUE)  # fair coin
```

```
## [1] "T" "H" "T" "T" "H" "T" "T" "T"
```

```
sample(c("H","T"), size = 8, replace = TRUE)  # fair coin
```

```
## [1] "T" "T" "H" "T" "H" "T" "H" "T"
```

Let's try a dice:

```
sample(1:6, size = 2, replace = TRUE, prob=c(3,3,3,4,4,4))
```

```
## [1] 5 3
```

```
sample(1:6, size = 2, replace = TRUE, prob=c(3,3,3,4,4,4))
```

```
## [1] 2 5
```

We use `replace=TRUE` to override the default sample without replacement. And, `prob=` to sample elements with different probabilities. The `set.seed()` function allow you to make a reproducible set of random numbers. Let's see the difference.

```
set.seed(123)
sample(c("H","T"), size = 8, replace = TRUE)
```

```
## [1] "H" "H" "H" "T" "H" "T" "T" "T"
```

```r
set.seed(123)
sample(c("H","T"), size = 8, replace = TRUE)
```

```
## [1] "H" "H" "H" "T" "H" "T" "T" "T"
```

How about data shuffle?

```r
x <- 1:12

sample(x)
```

```
##  [1] 11  5  4  6  1  2  3 12  9 10  8  7
```

```r
sample(x, replace = TRUE)
```

```
##  [1]  9  9  3  8 10  7 10  9  3  4  1 11
```

Here is an example: **let's generate 501 coin flips**. The true model will be that it should have heads half the time and tails half the time.

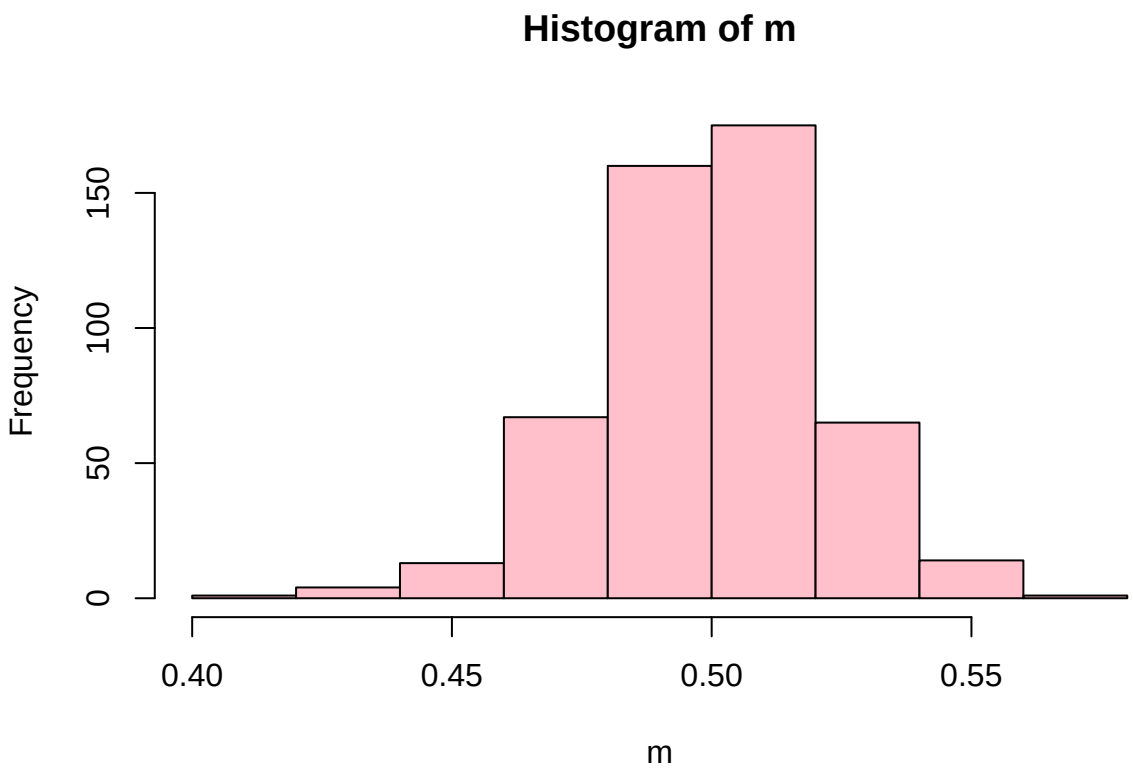
```r
coins <- sample(c("Heads","Tails"), 501, replace = TRUE)
mean(coins=='Heads')
```

```
## [1] 0.510978
```

It's not 50-50%. So what's our conclusion? Did this whole thing work or not?

What if it always skews on the same side? In other words, what if it's always bias towards heads or tails in every sample with 501 flips? We will do our first simulation to answer it:

```r
m <- c()

for (i in 1:500) {
  coins <- sample(c("Heads","Tails"), 501, replace = TRUE)
  m[i] <- mean(coins=='Heads')
}

hist(m, col="pink")
```




```
mean(m)
```

```
## [1] 0.4999082
```

```
sqrt(var(m))
```

```
## [1] 0.02157032
```

2. Random number generating with probablity distributions

Here are the common probability distributions in R. Search help for more detail.

`beta(shape1, shape2, ncp)`,
`binom(size, prob)`,
`chisq(df, ncp)`,
`exp(rate)`,
`gamma(shape, scale)`,
`logis(location, scale)`,
`norm(mean, sd)`,
`pois(lambda)`,
`t(df, ncp)`,
`unif(min, max)`,

`dxxx(x,)` returns the density or the value on the y-axis of a probability distribution for a discrete value of x,
`pxxx(q,)` returns the cumulative density function (CDF) or the area under the curve to the left of an x value on a probability distribution curve,
`qxxx(p,)` returns the quantile value, i.e. the standardized z value for x,
`rxxx(n,)` returns a random simulation of size n

```
rnorm(6) #  6 std nrml distribution values
```

```
## [1]  1.8975895  0.7850431 -0.7196878 -0.6208964 -1.6697667  1.3771364
```

```r
rnorm(10, mean = 50, sd = 19) # set parameters
```

```
##  [1] 71.084319 30.547257 59.607488 36.787610 83.653774 55.156698 45.361055
##  [8] 31.527189 76.652482  7.628797
```

```r
runif(n = 10, min = 0, max = 1) #uniform distribution
```

```
##  [1] 0.626936582 0.202141155 0.082219818 0.003339609 0.836393981 0.896037557
##  [7] 0.711370967 0.087794516 0.682685568 0.518216509
```

```r
rbinom(n = 8, size = 1, p = 0.5)
```

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

```r
# 18 trials, sample size 10, prob success =.2
rbinom(18, 10, 0.5)
```

```
##  [1] 3 3 7 7 6 6 5 6 7 1 7 6 4 7 6 7 4 5
```

```r
mean(rbinom(100, 10, 0.5))
```

```
## [1] 5.07
```

3. Simulation for statistical inference

Let's predict number of girls in 400 births, where probability of female birth is 48.8%

```r
n.girls <- rbinom(1, 400, 0.488)
n.girls
```

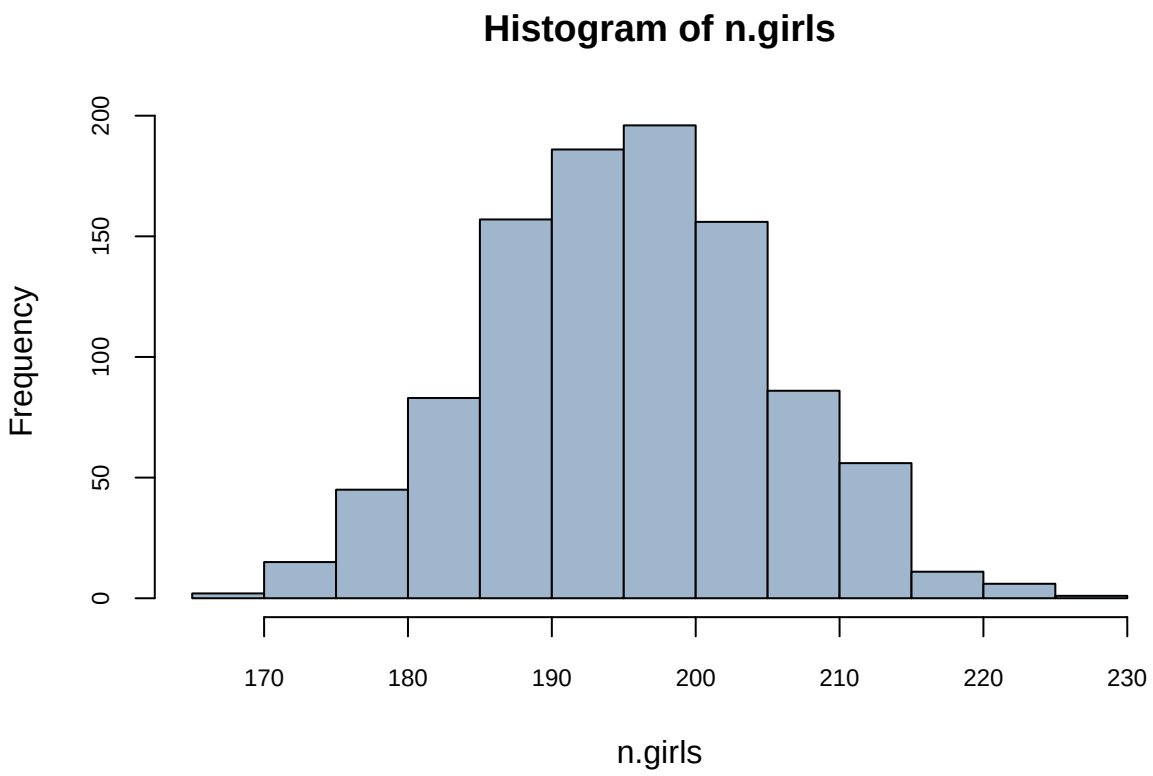
```
## [1] 189
```

```r
n.girls/400
```

```
## [1] 0.4725
```

Now, to get distribution of the simulations, repeat the simulation many times.

```r
n.sims <- 1000
n.girls <- rbinom(n.sims, 400, .488)
hist(n.girls, col = "slategray3",  cex.axis = 0.75)
```

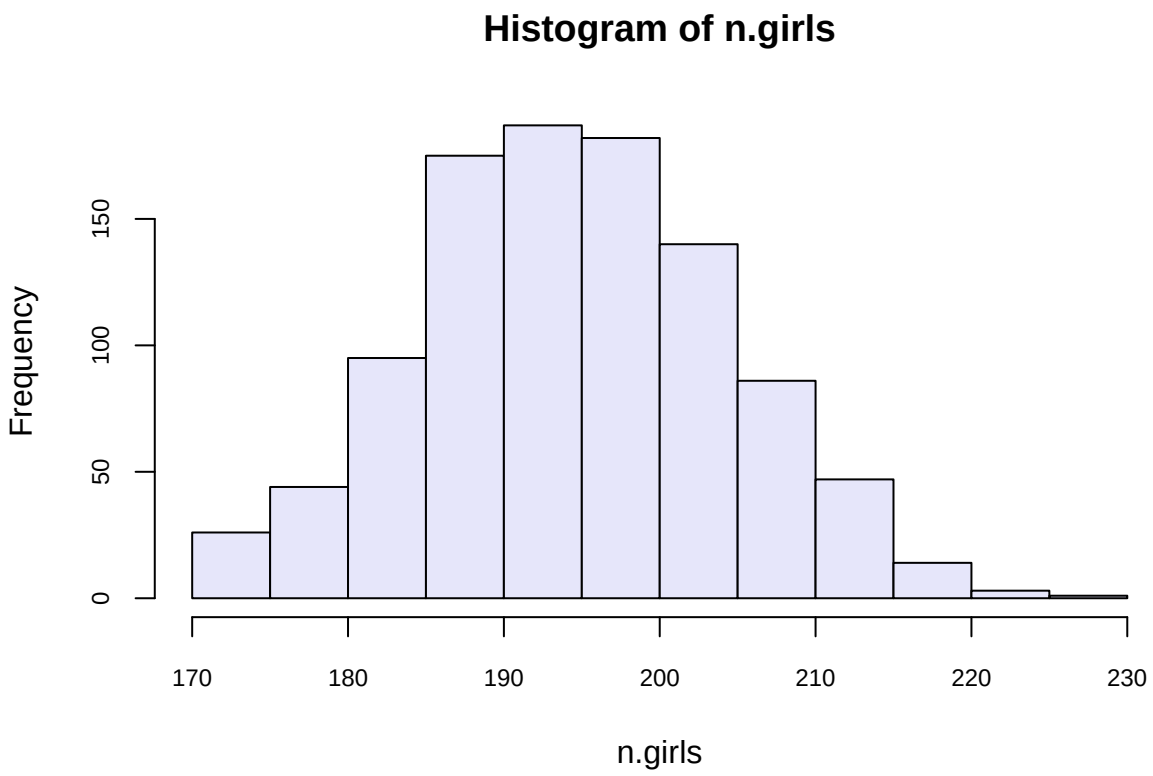



```
mean(n.girls)/400
```

```
## [1] 0.48953
```

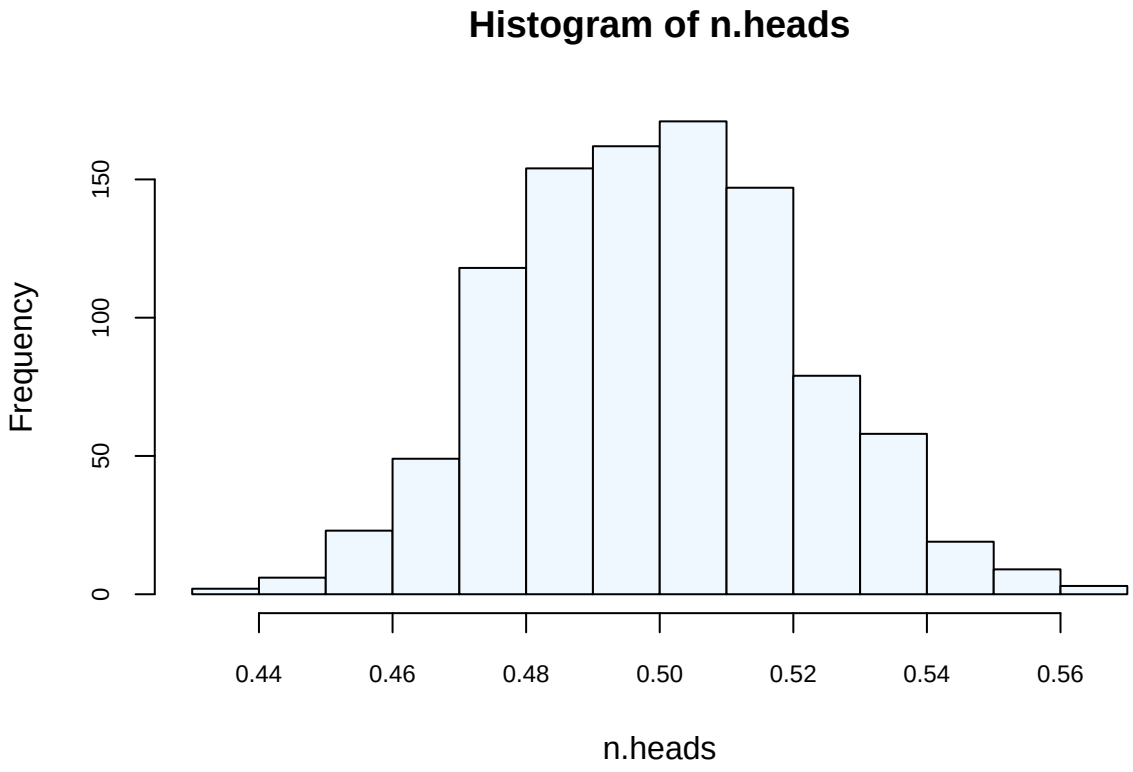
This is called as *sampling distribution*. Can we do same thing with a loop?

```
n.sims <- 1000
n.girls <- c() # create vector to store simulations

for (i in 1:n.sims){
n.girls[i] <- rbinom(1, 400, 0.488)
}

hist(n.girls, col = "lavender", cex.axis = 0.75)
```



Let's apply a similar simulation to our coin flipping?

```
n.sims <- 1000
n.heads <- c()

for (i in 1:n.sims){
n.heads[i] <- mean(rbinom(n = 501, size = 1, p = 0.5))
}

hist(n.heads, col="aliceblue", cex.axis = 0.75)
```




```
mean(n.heads)
```

```
## [1] 0.4992814
```

What's the 95% confidence interval for the mean?

```
sd <- sd(n.heads)
CI95 <- c(-2*sd+mean(n.heads), 2*sd+mean(n.heads))
CI95
```

```
## [1] 0.4556697 0.5428931
```

What happens if we use a "wrong" estimator for the mean, like `sum(heads)/300`?

```
n.sims <- 1000
n.heads <- c()

for (i in 1:n.sims){
n.heads[i] <- sum(rbinom(n = 501, size = 1, p = 0.5))/300
}
mean(n.heads)
```

```
## [1] 0.8342767
```

4. Cretaing data with a Data Generating Model (DGM)

One of the major tasks of statistics is to obtain information about populations. In most cases, the population is unknown and what is known for the researcher is a finite subset of observations drawn from this population.

The main aim of the statistical analysis is to obtain information from the population through the analysis of the sample. Since very few information is know about the population characteristics, one has to establish some assumptions about the behavior of this unknown quantity.

For example, for a regression analysis, we can state that the whole population regression function (PRF) as

a linear function of the different values of X. One important issue related to the PRF is the error term (u_i) in the regression equation. For a pair of realizations (x_i, y_i) from the random variables (X, Y), we can write the following equalities:

$$y_i = E\left(Y|X = x_i\right) + u_i = \alpha + \beta x_i + u_i$$

and

$$E\left(u|X = x_i\right) = 0$$

This result implies that for $X = x_i$, the divergences of all values of Y with respect to the conditional expectation $E(Y|X = x_i)$ are averaged out.

There are several reasons for the existence of the error term in the regression:

1. The error term takes into account variables that are not in the model;
2. We do not have a great confidence about the correctness of the model;
3. We do not know if there are measurement errors in the variables.

In a regression analysis, the PRF is a Data Generating Model for y_i, which is unknown to us so that we try to discover it from a sample, that is the only available data for us.

If we assume that there is a specific PRF that generates the data, then, given any estimator of α and β, namely $\hat{\beta}$ and $\hat{\alpha}$, we can estimate them from our sample by the sample regression function (SRF):

$$\hat{y}_i = \hat{\alpha} + \hat{\beta} x_i, \quad i = 1, \cdots, n$$

The relationship between the PRF and SRF is:

$$y_i = \hat{y}_i + \hat{u}_i, \quad i = 1, \cdots, n$$

where $\hat{u_i}$ is denoted the residuals from SRF.

With a data generating process (DGP) at hand, it is possible to create new simulated data. With α, β and the vector of exogenous variables X (fixed), a sample of size n can be used to obtain N values of Y with random variable u.

This yields one complete **population** of size N. Note that this artificially generated set of data could be viewed as an example of real-world data that a researcher would be faced with when dealing with the kind of estimation problem this model represents. Note especially that the set of data obtained depends crucially on the particular set of error terms drawn. A different set of error terms would create a different data set of Y for the same problem.

With the artificial data we generated, DGM is now known and the whole population is accesible. That is, we can test many models on different samples drawn from this population in order to see whether their inferencial properties are in line with DGM. We'll have several examples below.

Here is the DGM:

$$Y_i = \beta_1 + \beta_2 X_{2i} + \beta_3 X_{3i} + \beta_4 X_{2i} X_{3i} + \beta_5 X_{5i},$$

with the following coefficient vector: $\beta = (12, -0.7, 34, -0.17, 5.4)$. Moreover x_2 is binary variable with values of 0 and 1 and x_5 and x_3 are highly correlated with $\rho = 0.65$. When we add the error term, u, which is independently and identically (i.i.d) distributed with $N(0, 1)$, we can get the whole *population* of 10,000 observations. DGM plus the error term is called the data generating process (DGP)

```r
library(MASS)
library(stargazer)

N <- 10000
x_2 <- sample(c(0,1), N, replace = TRUE)

X_corr<- mvrnorm(N, mu = c(0,0), Sigma = matrix(c(1,0.65,0.65,1), ncol = 2),
                 empirical = TRUE)

#We can check their correlation
cor(X_corr)
```

```
##      [,1] [,2]
## [1,] 1.00 0.65
## [2,] 0.65 1.00
```

```r
#Each column is one of our variables
x_3 <- X_corr[,1]
x_5 <- X_corr[,2]

#interaction
x_23 <- x_2*x_3

# Now DGM
beta <- c(12, -0.7, 34, -0.17, 5.4)
dgm <- beta[1] + beta[2]*x_2 + beta[3]*x_3 + beta[4]*x_23 + beta[5]*x_5

#And our Yi
y <- dgm + rnorm(N,0,1)
pop <- data.frame(y, x_2, x_3, x_23, x_5)

stargazer(pop, type = "text", title = "Descriptive Statistics",
          digits = 1, out = "table1.text")
```

```
##
## Descriptive Statistics
## ==========================================
## Statistic   N    Mean  St. Dev.  Min    Max
## ------------------------------------------
## y         10,000 11.7    37.7   -147.7 172.8
## x_2       10,000  0.5    0.5      0      1
## x_3       10,000  0.0    1.0     -4.3   4.5
## x_23      10,000 0.002   0.7     -4.3   3.3
## x_5       10,000  0.0    1.0     -3.9   4.1
## ------------------------------------------
```

```r
#The table will be saved in the working directory
#with whatever name you write in the out option.
#You can open this file with any word processor
```

Now we are going to sample this population:

```r
n <- 500 #sample size
ind <- sample(nrow(pop), n, replace = FALSE)
sample <- pop[ind, ]
```

```r
str(sample)
```

```
## 'data.frame':    500 obs. of  5 variables:
##  $ y   : num  12.3 -15.6 -33.7 21.8 32 ...
##  $ x_2 : num  1 0 1 1 0 0 1 0 0 0 ...
##  $ x_3 : num  0.0494 -0.7378 -1.3929 0.2565 0.3375 ...
##  $ x_23: num  0.0494 0 -1.3929 0.2565 0 ...
##  $ x_5 : num  -0.126 -0.675 0.178 0.238 1.256 ...
```

and run a SRF:

```r
model <- lm(y ~ ., data = sample)
stargazer(model, type = "text", title = "G O O D -  M O D E L",
          dep.var.labels = "Y",
          digits = 3)
```

```
##
## G O O D - M O D E L
## ================================================
##                         Dependent variable:
##                     ---------------------------
##                                   Y
## ------------------------------------------------
## x_2                            -0.951***
##                                 (0.090)
##
## x_3                            33.951***
##                                 (0.075)
##
## x_23                            -0.056
##                                 (0.094)
##
## x_5                             5.425***
##                                 (0.056)
##
## Constant                       12.133***
##                                 (0.064)
##
## ------------------------------------------------
## Observations                      500
## R2                               0.999
## Adjusted R2                      0.999
## Residual Std. Error         1.003 (df = 495)
## F Statistic          161,981.400*** (df = 4; 495)
## ================================================
## Note:                *p<0.1; **p<0.05; ***p<0.01
```

As you can see the coefficents are very close to our "true" coefficents specified in DGM. Now we can test what happens if we omit x_5 in our SRF and estimate it? Let's see.

```r
n <- 500 #sample size
ind <- sample(nrow(pop), n, replace = FALSE)
sample <- pop[ind, ]
str(sample)
```

```
## 'data.frame':    500 obs. of  5 variables:
```

```
##  $ y   : num  -5.986 40.207 -39.487 -0.372 100.374 ...
##  $ x_2 : num  1 0 0 0 1 1 1 0 0 0 ...
##  $ x_3 : num  -0.617 0.724 -1.328 -0.266 2.434 ...
##  $ x_23: num  -0.617 0 0 0 2.434 ...
##  $ x_5 : num  0.572 0.702 -1.043 -0.83 1.119 ...
```

```
model_bad <- lm(y ~ x_2 + x_3 + x_23, data = sample)
stargazer(model_bad, type = "text", title = "B A D - M O D E L",
          dep.var.labels = "Y",
          digits = 3)
```

```
##
## B A D - M O D E L
## ================================================
##                         Dependent variable:
##                     ---------------------------
##                                   Y
## ------------------------------------------------
## x_2                              -0.177
##                                  (0.385)
##
## x_3                             37.808***
##                                  (0.270)
##
## x_23                             -0.478
##                                  (0.383)
##
## Constant                        11.489***
##                                  (0.276)
##
## ------------------------------------------------
## Observations                       500
## R2                                0.987
## Adjusted R2                       0.987
## Residual Std. Error         4.295 (df = 496)
## F Statistic           12,841.860*** (df = 3; 496)
## ================================================
## Note:                *p<0.1; **p<0.05; ***p<0.01
```

Now it seems that none of the coefficients are as good as before, except for the intercept. This is a so-called **omitted variable bias (OVB) problem**, also known as a model underfitting or specification error. Would it be the case that this is a problem for only one sample? We can simulate the results many times and see whether **on average** $\hat{\beta}_3$ is biased or not.

```
n.sims <- 500
n <- 500 #sample size
beta_3 <- c()

for (i in 1:n.sims){
  ind <- sample(nrow(pop), n, replace = FALSE)
  sample <- pop[ind, ]
  model_bad <- lm(y ~ x_2 + x_3 + x_23, data = sample)
  beta_3[i] <- model_bad$coefficients["x_3"]
}
summary(beta_3)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   36.70   37.31   37.47   37.48   37.65   38.39
```

As we can see the OVB problem is not a problem in one sample. We withdrew a sample and estimated the same underfitting model 500 times with a simulation. Therefore, we collected 500 $\widehat{\beta}_3$. The average is 37.47. If we do the same simulation with a model that is correctly specified, you can see the results: the average of 500 $\widehat{\beta}_3$ is 34, which is the "correct"true" coefficent in our DGM.

```
n.sims <- 500
n <- 500 #sample size
beta_3 <- c()

for (i in 1:n.sims){
  ind <- sample(nrow(pop), n, replace = FALSE)
  sample <- pop[ind, ]
  model_good <- lm(y ~ x_2 + x_3 + x_23 + x_5, data = sample)
  beta_3[i] <- model_good$coefficients["x_3"]
}
summary(beta_3)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   33.80   33.97   34.01   34.02   34.06   34.24
```

5. Bootstrapping

Bootstrapping is the process of resampling with replacement (all values in the sample have an equal probability of being selected, including multiple times, so a value could have a duplicate). Resample, calculate a statistic (e.g. the mean), repeat this hundreds or thousands of times and you are able to estimate a precise/accurate uncertainty of the mean (confidence interval) of the data's distribution. There are less assumptions about the underlying distribution using bootstrap compared to calculating the standard error directly.

Generally bootstrapping follows the same basic steps:

- Resample a given data set a specified number of times,
- Calculate a specific statistic from each sample,
- Find the standard deviation of the distribution of that statistic.

In the following bootstrapping example we would like to obtain a standard error for the estimate of the mean. See more about bootstrapping here.

Let's create the data set by taking 100 observations from a normal distribution with mean 5 and standard deviation 3:

```
n = 100
data <- rnorm(n, 5, 3) #rounding each observation to nearest integer
data[1:10]
```

```
##  [1]  4.6129355  1.0464058  3.9439638  1.3620474 -0.9884319  1.3843353
##  [7]  4.1782637  5.9852208  4.2574833  2.5494792
```

```
mean(data)
```

```
## [1] 4.942202
```

Here is the sampling with bootstrapping:

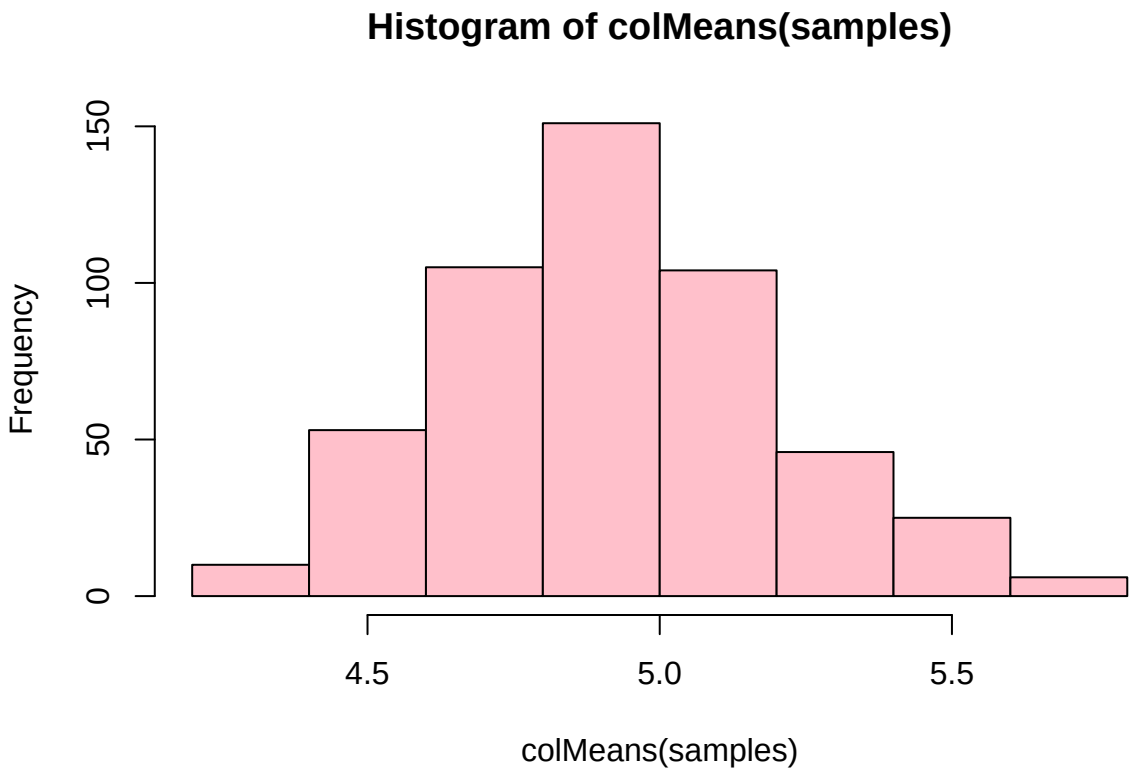
```
mcn = 500
samples <- matrix(0, nrow = n, ncol = mcn)
```

```r
for (i in 1:mcn) {
  samples[,i] <- sample(data, n, replace = TRUE)
}

#display the first of the bootstrap samples
samples[, 1]
```

```
##   [1]  2.0402919  2.6110022  0.2363332  5.7030987  5.4862931  7.2504436
##   [7] 12.2589600  3.1626715  7.0224688  8.2065323  3.9970989  3.7216881
##  [13]  1.7758114  8.7386478  3.8723150  7.9735552  0.9206556 11.3487940
##  [19]  1.1946056  8.2219281  1.1946056  9.1235670 10.9086025 11.3487940
##  [25]  6.3988719  0.8331259  4.3894023 10.3369357  4.2656415  7.1324534
##  [31]  6.8871781  4.2187148  9.1235670  4.7929694  9.4671082  4.5820745
##  [37]  1.0464058  3.2925998  2.5494792  0.2042058  5.1235440  1.1751642
##  [43]  1.3843353  7.0934082  4.1488158 10.3369357  3.2925998  4.2655317
##  [49] 10.9086025  3.3084743  1.5185165  8.3180628  4.4328547  7.2504436
##  [55]  7.2504436  5.0846523  1.3620474 11.3487940  3.2471192  5.3458731
##  [61]  3.7216881  6.3988719  4.1488158  1.1946056 10.2561830  7.1077383
##  [67]  3.8585821 11.7190326  1.7758114  2.6110022 -0.9884319  6.6794965
##  [73]  2.1344164  2.3801297  4.1488158  4.2655317  8.2219281 12.2589600
##  [79]  7.0224688  2.6110022  1.1751642  5.3907657  6.1175657  3.1626715
##  [85] 10.9086025  3.3084743  5.6038388  4.2574833  8.2219281  1.5185165
##  [91]  4.1782637  0.8269262  3.9970989  0.9206556  4.2656415  7.2327223
##  [97] 12.2589600  7.2327223  8.2065323  4.2574833
```

Calculating the mean for each bootstrap sample:

```r
hist(colMeans(samples), col = "pink")
```




```r
#and the mean of all means
```

```r
mean(colMeans(samples))
```

```
## [1] 4.927029
```

Calculating the standard deviation of the distribution of means:

```r
sdxbar <- sqrt(var(colMeans(samples)))
sddata <- sqrt(var(data))
sdxbar
```

```
## [1] 0.2787832
```

```r
sddata
```

```
## [1] 2.901173
```

Why $\mathbf{sd}(\bar{X})$ is different than $\mathbf{sd}(x_i)$? What's $\mathbf{Var}(\bar{X})$? With the assumption of i.i.d. it can be expressed as follows:

$$\mathbf{Var}(\bar{X}) = \mathbf{Var}\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right) = \frac{1}{n^2}\sum_{i=1}^{n}\mathbf{Var}(x_i) = \frac{1}{n^2}\sum_{i=1}^{n}\sigma^2 = \frac{1}{n^2}n\sigma^2 = \frac{\sigma^2}{n}.$$

We do not know σ^2 but we can approximate it by $\hat{\sigma}^2$, which is the variance of the sample.

$$\mathbf{Var}(\bar{X}) = \frac{\hat{\sigma}^2}{n} \quad \Rightarrow \quad \mathbf{se}(\bar{X}) = \frac{\hat{\sigma}}{\sqrt{n}}$$

Note that the terms, **standard deviation** and **standard error**, often lead to confusion about their interchangeability. We use the term standard error for the sampling distribution (standard error of the mean - SEM): the standard error measures how far the sample mean is likely to be from the population mean. Whereas the standard deviation of the sample (population) is the degree to which individuals within the sample (population) differ from the sample (population) mean.

In our case, since

$$\mathbf{sd}(\bar{X}) = \frac{\sigma}{\sqrt{n}} \quad \Rightarrow \quad \mathbf{sd}(\bar{X})\sqrt{n} = \sigma.$$

```r
sdxbar*sqrt(n)
```

```
## [1] 2.787832
```

```r
sddata
```

```
## [1] 2.901173
```

6. Monthy Hall

The Monty Hall problem is a brain teaser, in the form of a probability puzzle, loosely based on the American television game show Let's Make a Deal and named after its original host, Monty Hall. The problem was originally posed (and solved) in a letter by Steve Selvin to the American Statistician in 1975 (Selvin 1975a), (Selvin 1975b). It became famous as a question from a reader's letter quoted in Marilyn vos Savant's "Ask Marilyn" column in Parade magazine in 1990:

Suppose you're on a game show, and you're given the choice of three doors: Behind one door is a car; behind the others, goats. You pick a door, say No. 1, and the host, who knows what's behind the doors, opens another door, say No. 3, which has a goat. He then says to you, "Do you want to pick door No. 2?" Is it to your advantage to switch your choice?

Vos Savant's response was that the contestant should switch to the other door (vos Savant 1990a). Under the standard assumptions, contestants who switch have a 2/3 chance of winning the car, while contestants who stick to their initial choice have only a 1/3 chance.

Many readers of vos Savant's column refused to believe switching is beneficial despite her explanation. After the problem appeared in Parade, approximately 10,000 readers, **including nearly 1,000 with PhDs**, wrote to the magazine, most of them claiming vos Savant was wrong. Even when given explanations, simulations, and formal mathematical proofs, many people still do not accept that switching is the best strategy. **Paul Erdős, one of the most prolific mathematicians in history, remained unconvinced until he was shown a computer simulation demonstrating the predicted result**.

The given probabilities depend on specific assumptions about how the host and contestant choose their doors. A key insight is that, under these standard conditions, there is more information about doors 2 and 3 that was not available at the beginning of the game, when door 1 was chosen by the player: the host's deliberate action adds value to the door he did not choose to eliminate, but not to the one chosen by the contestant originally. Another insight is that switching doors is a different action than choosing between the two remaining doors at random, as the first action uses the previous information and the latter does not. Other possible behaviors than the one described can reveal different additional information, or none at all, and yield different probabilities.

Here is the simple Bayes rule: $Pr(A|B) = Pr(B|A)Pr(A)/Pr(B)$**.**

Let's play it: The player picks Door 1, Monty Hall opens Door 3. My question is this:

$Pr(CAR = 1|Open = 3) < Pr(CAR = 2|Open = 3)$?

If this is true the player should always switch. Here is the Bayesian answer:

$Pr(Car = 1|Open = 3) = Pr(Open = 3|Car = 1)Pr(Car = 1)/Pr(Open = 3) = 1/2$ x $(1/3)$ / $(1/2) = 1/3$

Let's see each number. Given that the player picks Door 1, if the car is behind Door 1, Monty should be indifferent between opening Doors 2 and 3. So the first term is 1/2. The second term is easy: Probability that the car is behind Door 1 is 1/3. The third term is also simple and usualy overlooked. This is not a conditional probability. If the car were behind Door 2, the probability that Monty opens Door 3 would be 1. And this explains why the second option is different, below:

$Pr(Car = 2|Open = 3) = Pr(Open = 3|Car = 2)Pr(Car = 2)/Pr(Open = 3) = 1$ x $(1/3)$ / $(1/2) = 2/3$

Figure 1: Image taken from http://media.graytvinc.com/images/690*388/mon+tyhall.jpg

Simulation to prove it

```
n <- 100000
```

Step 1: Decide the number of plays

Step 2: Define all possible door combinations 3 doors, the first one has the car. All possible outcomes for the game:

```
outcomes <- c(123,132,213,231,312,321)
```

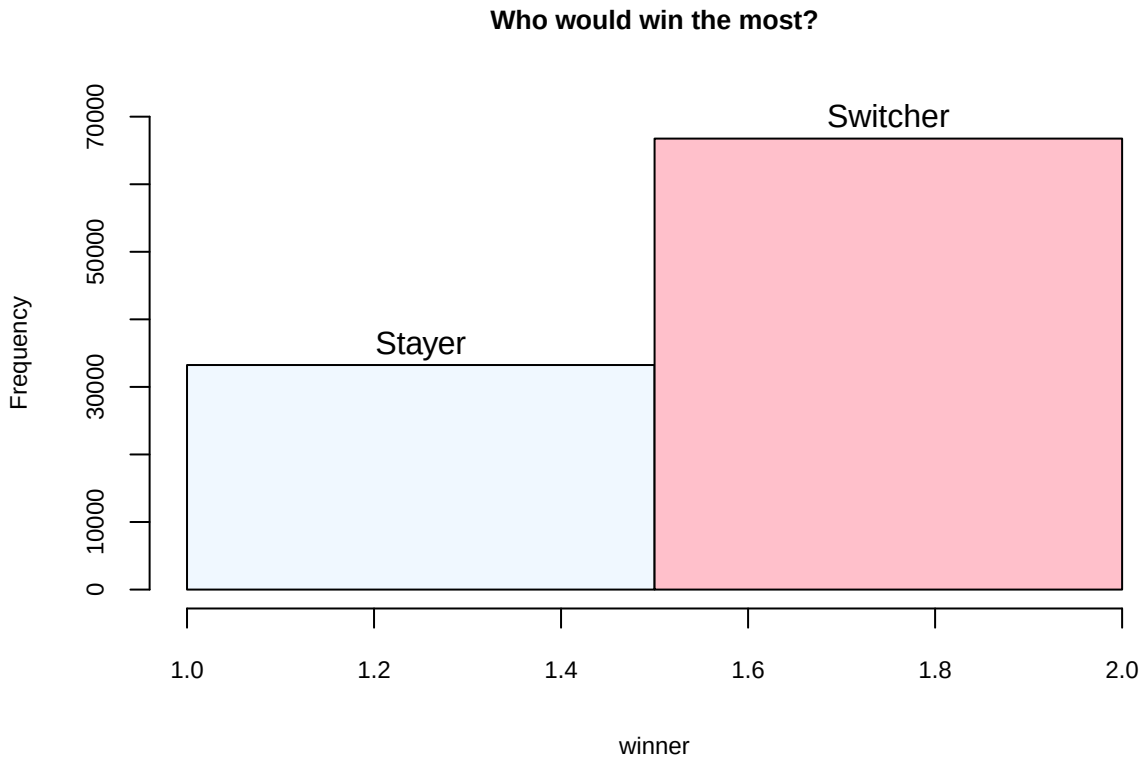
```
car <- c()
goat1 <- c()
goat2 <- c()
choice <- c()
monty <- c()
winner <- c()
```

Step 3: Create empty containers where you store the outcomes from each game

```
for (i in 1:n){
  doors <- sample(outcomes,1) #The game's door combination
  car[i] <- substring(doors, first = c(1,2,3), last = c(1,2,3))[1] #the right door
  goat1[i] <- substring(doors, first = c(1,2,3), last = c(1,2,3))[2] #The first wrong door
  goat2[i] <- substring(doors, first = c(1,2,3), last = c(1,2,3))[3] #The second wrong door

  #Person selects a random door
  choice[i] <- sample(1:3,1)

  #Now Monty opens a door
  if (choice[i] == car[i])
    {monty[i] = sample(c(goat1[i],goat2[i]),1)}
  else if (choice[i] == goat1[i])
    {monty[i] = goat2[i]}
  else
    {monty[i] = goat1[i]}

  # 1 represents the stayer who remains by her initial choice
  # 2 represents the switcher who changes her initial choice
  ifelse(choice[i]==car[i], winner[i] <- 1, winner[i] <- 2)
}
```

Step 4: Loop

```
hist(winner, breaks = 2, main = "Who would win the most?",
     ylim = c(0,70000), labels = c("Stayer", "Switcher"),
     col = c("aliceblue", "pink"),
     cex.axis = 0.75, cex.lab = 0.75, cex.main = 0.85)
```



Step 5: Chart

The simulation is inspired by mrajter