TP2 ANEXO: Muestreo 2.0

En esta segunda parte de la práctica vamos a utilizar un programa estadístico (R + Rstudio) para simular distintos casos y profundizar en el concepto de muestreo. Para ello, vamos a necesitar primero aprender algunos conceptos básicos del programa, algunos conceptos básicos de probabilidad y estadística, y algunos conceptos básicos de programación.

1. Instalar R y Rstudio

R es un lenguaje de programación específicamente diseñado para problemas estadísticos. Rstudio es un programa que corre sobre R y que permite una visualización más amigable. Instalar ambos programas es bastante sencillo. Para instalar R:

Diríjase en un navegador de internet a:

http://mirror.fcaglp.unlp.edu.ar/CRAN/

Seleccione su sistema operativo (por ejemplo, windows) y en la siguiente página seleccione “Install R for the first time” para instalar los paquetes mínimos necesarios

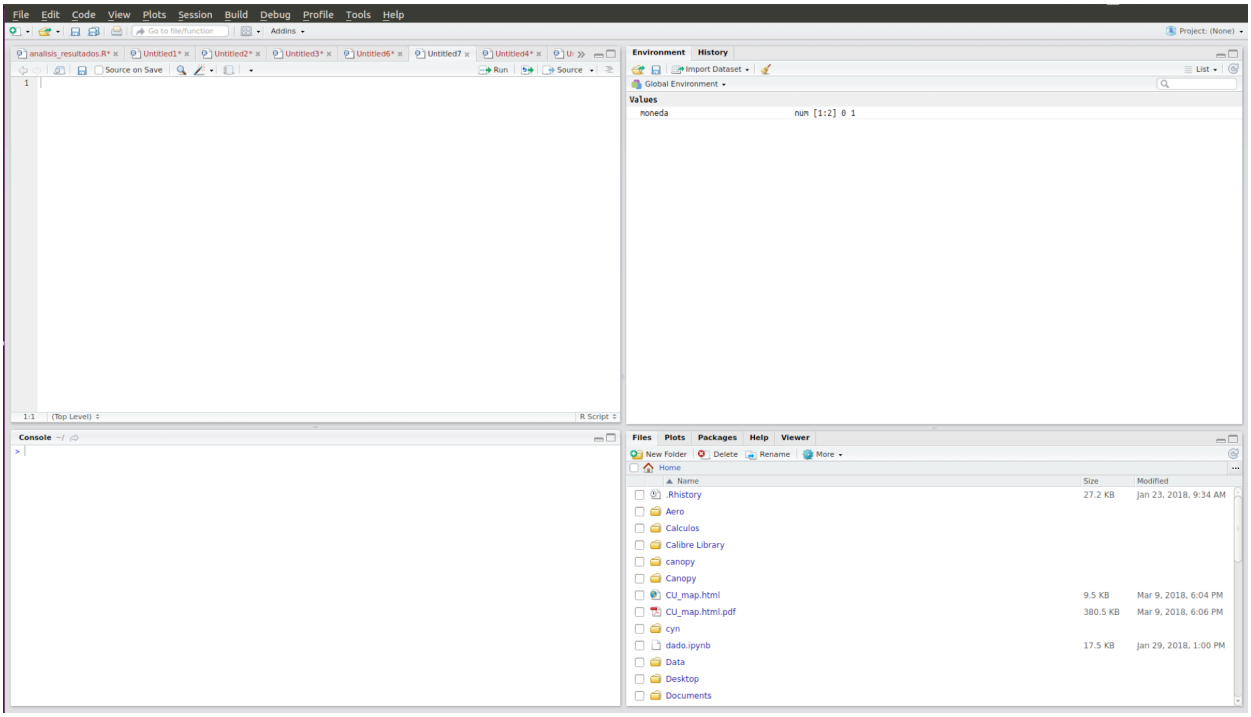

Figura 1. Pantalla del RStudio. Izquierda arriba: Editor. Izquierda Abajo: Consola. Derecha Arriba: visualizador de variables. Derecha Abajo: Explorador, instalador de paquetes, visualizador de gráficos, ayuda y otros.

Ejecute el instalador y siga las instrucciones. Para instalar Rstudio, diríjase a https://www.rstudio.com/products/rstudio/download/ y descargue la versión libre de rstudio desktop. Ejecute el instalador y siga las instrucciones.

2. Algunas cosas básicas de R y Rstudio

Hay dos formas básicas de comunicarse con R: mediante la consola, escribiendo comando por comando o mediante un "script", encadenando una serie de comandos y teniendo la posibilidad de guardarlos y ejecutarlos cuando queramos.

Empecemos escribiendo en la consola (>) el siguiente comando (el mayor no se escribe!):

```
> moneda <- c('cara', 'ceca')
```

No se debería ver un resultado en la consola, pero debería aparecer una variable llamada moneda en el visor de variables arriba a la derecha. ¿Cómo se lee el comando?

El símbolo "**<-**" significa asignar y el comando **c()** significa combinar. El comando se leería en "humano": combinar la palabra "cara" y la palabra "ceca" en un vector y asignarlo a la variable "moneda". En el contexto que estamos trabajando, una variable es una caja en la que la computadora guarda un valor.

Si quisiésemos ver que hay en moneda, escribimos en la consola:

```
> moneda
```

y como resultado obtenemos:

```
[1] "cara" "ceca"
```

Ahora que tenemos una moneda, "lancémosla" al aire:

```
> sample(moneda, 1)
```

El resultado puede ser "cara" o "ceca". ¿Qué pasa si lanzamos la moneda 10 veces? Dado que moneda solo tiene dos valores, la probabilidad de obtener cualquiera de ellos es de 0.5 y el valor "esperado" es de 5 caras (y 5 cecas). Veamos que ocurre:

```
> sample(moneda, 10, replace = TRUE)
```

¿Qué obtuvo? Si El parámetro **replace = TRUE**, le dice a la función **sample** que el muestreo se hace con reposición. Para simular la moneda, la computadora pone los dos probables valores en una "bolsita" y saca sin mirar un valor. Si no repusiera el valor que sacó de la bolsita, solo podría muestrear dos veces (y ya sabríamos el segundo valor con sólo sacar el primero. ¿Porqué?).

En R podemos tener variables representando escalares, vectores y matrices. Se puede acceder a valores en particular de los vectores y las matrices. A este proceso se le conoce como "slicing". Por ejemplo, los siguientes comandos generan una matriz de 3x3 con números al azar entre 0 y 1:

```
> set.seed(42)
> m <- matrix(runif(9), ncol = 3, nrow = 3)
```

Si queremos extraer la primer columna de la matriz "m", podemos escribir:

```
> m[,1]
```

Por otro lado, si queremos extraer de la tercer columna de la matriz “m”, el segundo valor, podemos escribir:

```
> m[2,3]
```

Por último, ante cualquier duda puede escribir **help**(comando), y se abrirá una ventana explicando el uso del mismo. Y si con eso no es suficiente, existe una enorme comunidad en la internet. Googlee.

3. Algunas cosas básicas de programación

A lo largo de este práctico vamos a usar algunas palabras a las que estamos acostumbrados usar de otra forma. Por ejemplo, en este práctico una variable es simplemente una etiqueta para guardar un valor (o conjunto de valores como un vector o una matriz). Por ejemplo, moneda es una variable que guarda un vector con dos variables.

Una función es un objeto que nos devuelve un valor (o valores) al ser aplicado sobre una variable. Por ejemplo, la función sample funciona así:

sample(variable, cantidad de veces, replace = TRUE o FALSE?)

dada una variable y un número de veces, la función devuelve valores muestreados de la variable. Cuantos? La cantidad de veces que le pidamos. Por último, algunas funciones tienen parámetros que modifican como se comportan. En este caso, dependiendo del valor de replace, sample devolverá distintos valores.

Por último, vamos a conocer una estructura de programación. Una de las ventajas de trabajar con computadoras es que pueden hacer las cosas muchas veces y mucho más rápido que nosotros. Sobre todo si son tareas repetitivas. Si queremos hacer algo N veces, podemos usar la estructura de “loop”. En R se puede escribir:

for (condición){comandos a repetir}

En otras palabras, R repite los comandos entre llaves mientras se cumpla la condición entre paréntesis. Por ejemplo:

```
> for(i in seq(1:10)){print(i)}
```

imprime los números de 1 a 10. En “humano” se lee: para i en la secuencia 1 a 10, repetir imprimir i. En otras palabras, i es una variable que va cambiando de valor de 1 a 10, y para cada valor la función print escribe el valor de i. En el momento de iniciar el loop, a i se le asigna el valor 1 (el primer valor de seq(1:10)). Luego el programa entra en las llaves y ejecuta lo que dice. Cuando llega a la llave final, cambia el valor de i y entra nuevamente en las llaves. Cuando i llega a 10, el programa hace la última vuelta por las llaves.

4. Simulando el muestreo de las bolitas

El objetivo de esta parte del TP, es utilizar lo que acabamos de aprender para realizar un “conteo” de pelotitas más eficiente. Por otro lado, podemos “fabricar” muestras más grandes sin tener que contar a mano millones de pelotitas y evaluar los resultados mucho más rápidamente.

Vamos a fabricar una caja con 600 pelotitas, de las cuales 12 van a ser el analito que queremos determinar. En otras palabras, la fracción x_{azul} es 0.02. Vamos a usar el comando **rep**, para replicar un vector la cantidad de veces que queramos. Por ejemplo, si queremos replicar 588 veces el valor "0", podemos escribir:

```
> rep(0, 588)
```

Usando los comandos **rep** y **c**, vamos a construir una caja que contenga 588 ceros y 12 unos (los unos representan nuestro analito, los 0 todo lo demás). Escriba los siguientes comandos en un script (a medida que los escriba, traté de leer lo que está haciendo y pregúntese por que lo estamos haciendo tan difícil!):

```
set.seed(42)

bolitas <- 600
x_azul <- 0.02
bolitas_azules <- floor(bolitas*x_azul)
bolitas_no_azules <- bolitas - bolitas_azules

caja <- c(rep(0, bolitas_no_azules), rep(1, bolitas_azules))
```

Sea creativo, y agregue un comando debajo del último para muestrear 10 veces de la caja sin reposición (pista: recuerde la moneda!). Ejecute cada uno de los comandos y chequee que el resultado es:

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

La conclusión es que creamos una caja con 600 bolitas (incluyendo 12 azules) y no hay bolitas azules!? Claramente hay algo más que nos está faltando. Lo que acaba de ocurrir es que una muestra de 10 bolitas no resulta suficiente para poder asegurar el valor de las 12 azules en 600. Repitamos el muestreo 10 veces y promediemos la cantidad de bolitas azules que aparecen. Para ello vamos a usar la estructura de programación "loop" que vimos en la parte anterior:

```
#Construyo una variable que muestrea i veces una cantidad k de bolitas
set.seed(42)
muestra <- NULL
k <- 10

for(i in seq(1:10)){

        #fraccion de bolitas azules que hay en la muestra
        n_azules <- mean(sample(caja, k, replace = FALSE))

        #concateno el resultado en muestra
        muestra <- c(muestra, n_azules)
}
```

Ejecute esta parte del script y chequee que su resultado sea:

```
[1] 0.0 0.1 0.0 0.0 0.0 0.1 0.0 0.0 0.0 0.0
```

¿Qué significa este resultado? Que si muestreamos 10 veces 10 pelotitas de la caja, vamos a ver sólo en dos ocasiones 1 pelotita azul. Calcule el promedio y desvío estándar de la muestra usando las funciones **mean()** y **sd()**. (Respuesta: 0.02 y 0.04)

El valor que encontramos de la fracción de bolitas azules es correcto. Pero el desvío que tenemos es enorme! Cuando hicimos el experimento contando bolitas de verdad, usamos vasos de precipitados de tamaño creciente. Los tamaños de muestreo son aproximadamente 40, 80 y 140 bolitas. Repita el código anterior muestreando 10 veces con esta cantidad de bolitas y obtenga la fracción de bolitas azules para cada tamaño de muestreo y su respectivo desvío estándar relativo porcentual. (Respuesta: 0.0175, 117%; 0.0175, 69%; 0.024, 42%)

Como se puede observar, la variabilidad al muestrear solo 10 veces es muy grande. ¿Podemos mejorar nuestra estimación aumentando la cantidad de muestreos? Repita el caso de 140 bolitas para 100 y 1000 veces. (Respuesta: 0.021, 50%; 0.021, 51%). Esta alta variabilidad está asociada a la poca cantidad de partículas que tenemos (pese a que estamos muestreando casi la tercera parte!) y va mejorar ni siquiera aumentando el número de repeticiones. Grafique los histogramas usando la función hist().

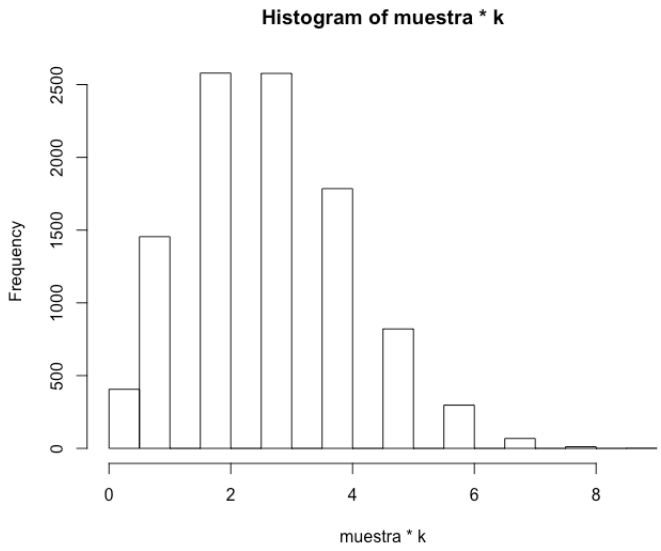

Figura 2. Histograma de # de bolitas azules en 10000 muestras de 140 bolitas, tomadas de la poblacion de 600.

Ahora hagamos el siguiente experimento mental. Supongamos que es posible "moler" las bolitas de colores, hasta el punto de que por cada bolita, obtenemos 100000 bolitas! Si partimos de 600 bolitas, ahora tenemos N = $6x10^8$ bolitas. Muestreemos 20 veces las siguientes cantidades de bolitas $Nx10^{-5}$, $Nx10^{-4}$, $Nx10^{-3}$ y $Nx10^{-2}$ (O lo que es lo mismo, muestreamos desde el 0.001% al 1% de la muestra). Para hacer las cosas más sencillas, abajo tiene un script en el que se automatiza el proceso. Léalo y trate de entender que es lo que estamos haciendo:

```
#Semilla random para que todos tengamos los mismos resultados
set.seed(42)

#Cantidad total de bolitas
bolitas <- 600*100000

#Fraccion de bolitas azules
x_azul <- 0.02
bolitas_azules <- floor(bolitas*x_azul)
bolitas_no_azules <- bolitas - bolitas_azules
```

```
#Construyo la caja con todas las bolitas
caja <- c(rep(0, bolitas_no_azules), rep(1, bolitas_azules))

#Inicializo la matriz que va a guardar los valores de las muestras
muestras <- NULL

#Numero de bolitas que voy a muestrear en cada caso y cantidad de repeticiones
tot <- floor(c(1e-5, 1e-4, 1e-3, 1e-2)*bolitas)
repeticiones <- 20

#Loop sobre la cantidad de bolitas a muestrear (esto se ejecuta 4 veces)
for (k in tot){

        #vacío el valor de muestra
        muestra <- NULL

        #Loop sobre las repeticiones (esto se ejecuta 20 veces)
        for(i in seq(1:repeticiones)){
                #Fraccion de bolitas azules que hay en la muestra
                n_azules <- mean(sample(caja, k, replace = FALSE))
                #Concateno el resultado en muestra
                muestra <- c(muestra, n_azules)
        }
        #Armo una matriz con los promedios de las repeticiones en cada columna
        muestras <- cbind(muestras, muestra)
}
#Le ponemos nombre a las columnas
colnames(muestras) <- tot

#Armamos una matriz con la cantidad de bolitas muestreadas por repeticion
vals <- matrix(rep(tot,repeticiones), nrow = repeticiones, ncol = length(tot),
byrow = TRUE)

#Graficamos cada repeticion en funcion bolitas muestreadas en escala logaritmica
plot(vals, muestras,log = "x", ylab = "Fraccion de bolitas azules", xlab = "# de
bolitas muestreadas")
desvio <- apply(muestras, MARGIN = 2, sd)
lines(tot,x_azul+desvio, col = "red")
lines(tot,x_azul-desvio, col = "red")
abline(h = x_azul, lty = 2, col = "red")
```

¿Qué se puede observar en el gráfico? Tratemos de racionalizar los experimentos que acabamos de hacer en función de la distribución binomial.

5. Un poco de teoría para chequear las simulaciones

Recuerde que si usted tiene n_A partículas A y n_B partículas B en una mezcla:

probabilidad de extraer A: $p = n_A/(n_A + n_B)$ (Eq.1a)
probabilidad de extraer B: $q = n_B/(n_A + n_B)$ (Eq.1b)

Si toma n partículas al azar, el valor esperado de partículas A es np y la varianza es $s^2 = npq$. Por lo tanto, el desvío estándar relativo es:

$$s_r = (q/np)^{1/2} \qquad \text{(Eq.2)}$$

La varianza relativa es:

$$R^2 = (s_r)^2 = q/pn \qquad \text{(Eq.3)}$$

Reacomodando llegamos a:

$$nR^2 = q/p \qquad \text{(Eq.4)}$$

La varianza relativa teórica la podemos obtener fácilmente:

```
varianza_relativa_teorica <- (1 - x_azul)/(x_azul * tot)
```

La varianza relativa simulada es un poco más engorrosa de escribir, pero fácil de obtener. Trate de pensar como obtenerla. Si no le sale, escriba:

```
varianza_relativa_simulada <- (apply(muestras*vals, MARGIN = 2,
sd)/apply(muestras*vals, MARGIN = 2, mean))^2
```

Grafique la varianza simulada versus la teórica usando los siguientes comandos:

```
plot(varianza_relativa_teorica, varianza_relativa_simulada, log = "xy", xlab = "Sr
teorica", ylab = "Sr simulada")
abline(a = 0, b = 1, col = "red")
```

Sabiendo la proporción de partículas azules, ¿Qué cantidad de partículas se deben muestrear para asegurar un S_r de 0.01 (o sea 1%)? ¿El resultado está de acuerdo a las simulaciones? En base a (Eq.4), muestre que se necesitan al menos 490000 partículas para muestrear las partículas azules en una población con x_{azul} = 0.02. ¿Qué le sugiere este resultado respecto a la imposibilidad de bajar el Sr en el experimento contando físicamente las bolitas?