CSCI E-63C: Week 1 Problem Set

Contents

Preface

This problem set, albeit not particularly sophisticated conceptually or technically, is designed to achieve two goals:

- make sure that you have correctly setup computational infrastructure (i.e. you can knit HTML representation of your results in R studio, which includes writing code that performs necessary computations, executing it, including its results in graphical and numerical form into Rmarkdown document and converting it to HTML format)
- jump start/refresh your statistical intuition (by drawing random samples from known distributions, estimating, plotting and describing properties of those samples over multiple random trials, etc).

To that end, this problem set is presented in the form of Rmarkdown file that can be opened and immediately knitted to HTML in Rstudio. The resulting HTML file is also provided. The starter code also includes an example of drawing random samples, repeatedly, from normal distribution and plotting resulting sample distributions as histograms using two frequently used graphical facilities in R: standard plots and `ggplot2` package.

One way to get started on this problem set is to open this Rmarkdown (.Rmd) file in Rstudio and "Knit HTML" from it (there is a button!). Then, by modifying Rmarkdown source and recompiling the HTML, you can see how the changes you make impact the resulting output. At this point you are totally empowered to do what is necessary to complete this problem set as described below.

Your submission (for this, as well as for all other upcoming weekly problem sets) should *always* include two files:

1. Working Rmarkdown (.Rmd) source, so that we can:
    - see the code that generated the results in its entirety
    - execute it independently if we need to see how it works or to debug it for better understanding what went wrong with your solution.
2. HTML representation of the results generated using "Knit HTML" from the above Rmarkdown source
    - this allows us to evaluate your final product without having to execute the source unless we absolutely have to

The ability to see your code and final results without having to re-run all the computations allows to decrease our collective dependency on concordance of our setups in terms of file names, locations, etc. that is not the goal of this class in itself. Try to think "portability", but don't overdo it: it helps to keep each weekly problem set as a separate folder/Rstudio project with all necessary data files in it, include all R functions you write for a given problem set in the same Rmarkdown file (as opposed to sourcing separate files which are easier to forget to add to the submission), use the same original input data files as provided in the problem sets (as opposed to manually transforming/reformatting them in a text editor or excel), etc. However you do not need to spend your time on writing super-portable code capable of detecting and automatically installing missing R packages, for example – as long as they are standard, we can download them when necessary.

Please note that while it is possible to show only the results in your HTML output and turn off display of the code that generated it, we kindly ask you to *never* do so for this course. You might indeed want to do that when you disseminate the results to your colleagues who are not interested in the code itself, but as it was explained above we prefer to see the code in its entirety for the purposes of grading, without the need to go back to the source RMD file every time.

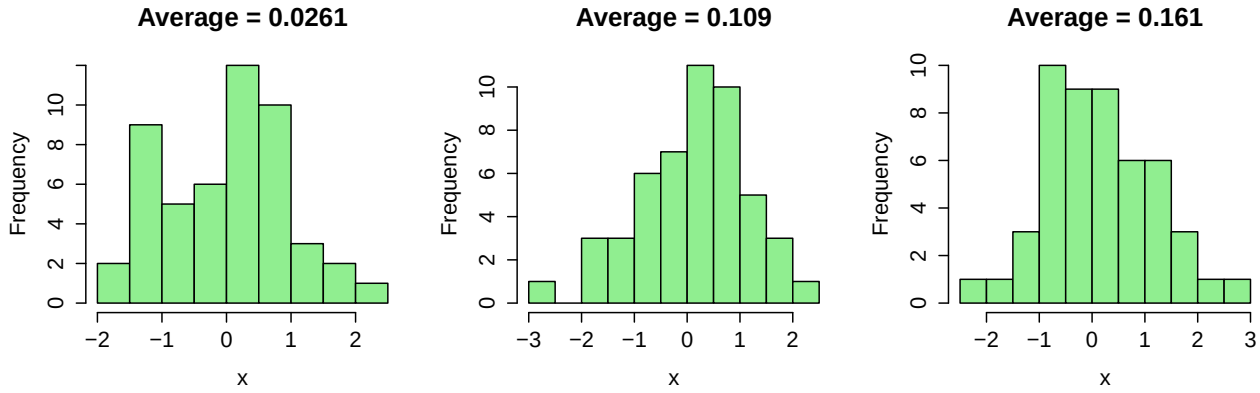
Finally, we'd like to remind you that whenever the homework assignment RMD file contains a preface (like this one, and any future assignment as we move forward), you are free to either keep it or to delete it (you may reuse or modify to your liking any starter code/examples we provide, of course, this is *not* considered plagiarism and goes without saying). Same applies to formulations of individual problems in the assignment, keep them or delete them. However, please make sure that the solutions to each problem are clearly marked (use separate sections), and whenever you are asked to discuss the obtained results please make sure that the discussion is clearly visible (font, color, separate subsection - anything). If you keep the whole original formulation of the problem in your document and just add couple of sentences at the end, let alone in the middle, of some paragraph - we may certainly miss it!

```
library(ggplot2)
```

A simple example

By the way of showing a simple working example of including the code and results it generates (plots in this case), here are histograms similar to those shown in lecture slides:

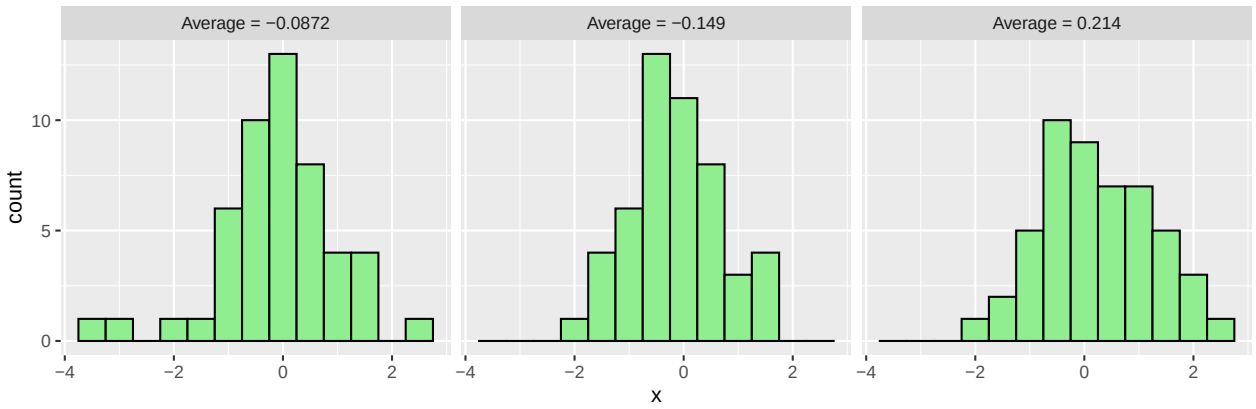
```
old.par <- par(mfrow=c(1,3),ps=16)
for (iTry in 1: 3) {
  x <- rnorm(50, mean=0, sd=1)
  hist(x, breaks=10, col='lightgreen', main=paste("Average =",signif(mean(x),3)))
}
```




```
par(old.par)
```

The code shown above runs a loop to draw three independent samples of size 50 from the standard normal distribution (zero mean, unit variance) and plots histograms of the obtained samples side by side.

Here is an example of accomplishing something very similar using `ggplot` (while precluding R code from showing up in HTML output, which, we remind you, we ask you NOT to do for the problem solution code you submit):



The choice of which plotting framework to use is totally yours – we often prefer to be opportunistic and use whichever one is the most expedient for doing what we are aiming to accomplish. We will be showing more examples as we move forward, just remember that while you can use both frameworks (and others - there are more than two!) in a single document, you cannot mix them in a single plot, they won't work together. You have to generate every single plot using either R base graphics or ggplot exclusively.

Problem 1 (30 points).

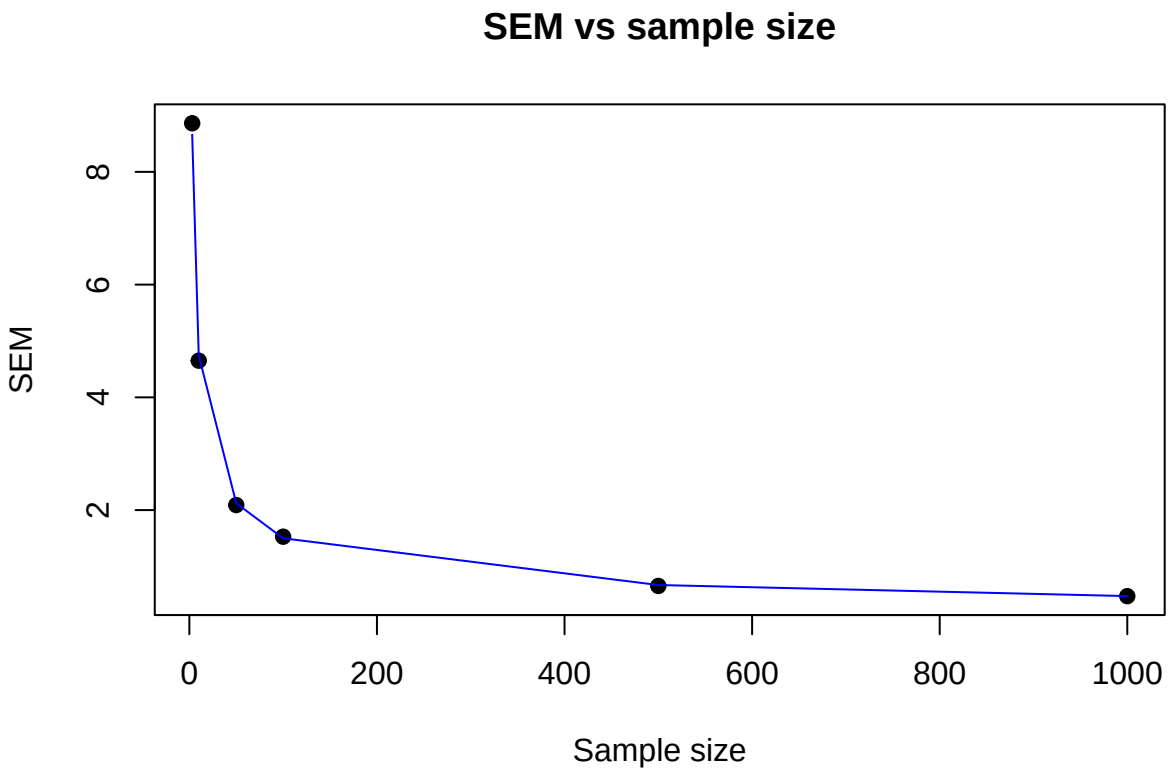
In class we have developed a simple simulation, in which we were looking at the mean of a sample as a random variable. Specifically, we were repeatedly drawing samples of size $N = 20$ from the same underlying normal distribution. In order to observe how the sample mean fluctuates from one experiment to the next we have simply plotted a histogram of the obtained mean values. In this problem, we will characterize the distribution of those sample means with its standard deviation AND examine how the spread of that distribution (i.e. the distribution of the *sample means*) decreases with increasing sample size, in line with quite intuitive notion that if we draw a larger sample, then its mean is expected to be closer, at least on average, to the true mean of the underlying population the sample was drawn from. The skeleton of the R code is presented below in the RMD document. Notice that its evaluation is turned off by `eval=FALSE` code chunk parameter because it is incomplete and will fail otherwise – once you modified the code so that it works, turn it to `eval=TRUE` (which is the default), so that it gets executed when you "Knit HTML":

```
# different sample sizes we are going to try:
sample.sizes <- c(3, 10, 50, 100, 500, 1000)

# we will use the vector below to save the standard deviations of the
# *distribution of the means* at each given sample size.
# Note that it's ok to initialize with an empty vector of length 0 - if we index
# it out-of-bounds while assigning a value later, the vector will autoexpand
# on assignment, see examples in the slides)
mean.sds <- numeric(0)
sigma <- 15

for ( N in sample.sizes ) { # try different sample sizes

 # INSERT YOUR CODE HERE: (you may want to check the slides).

 # 1) At each given N (i.e. in each iteration of the outer loop) you have to draw large number
 # (e.g. 1000) of samples of that size N, from the distribution of your choice (e.g. normal,
 # uniform, exponential, ...), and calculate the mean of *each* of those samples.
 # Save all those means into a vector 'm'.
```

```
 m <- replicate(1000, mean(rnorm(n = N, mean = 100, sd = sigma)))
 # 2) Now, with vector m in hand, we want to characterize how much the sample mean fluctuates
 # from one experiment (experiment=drawing a sample of N measurements) to the next. Instead of just
 # plotting a histogram as we did in class, this time we will calculate the standard
 # deviation of the distribution represented by the vector m. Use function sd() to achieve that.
 sd(m)
 # 3) save the result (sd of the distributions of the means for current N) into the vector mean.sds
 # defined above. You can use c() to concatenate, or you can use an indexing variable;
 # in the latter case you will need to add it to the code and increment properly
 # mean.sds <- append(mean.sds, c(sd(m)), after = length(mean.sds))
 mean.sds <- c(mean.sds, sd(m))
}

# at this point, you should have the vector mean.sds filled. It should have length 6 and keep the
# calculated values of the standard deviation of the mean (known as the standard error of the mean, SEM)
# at different sample sizes: mean.sds[1] is the SEM at N=3, mean.sds[2] is the SEM at N=10, and so on.
length(mean.sds)
```

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

```
# let us now plot the SEM (i.e. the "typical" error we expect the sample mean to exhibit in any
# given experiment) as a function of the sample size, N.

plot(sample.sizes, mean.sds, main = "SEM vs sample size",
     pch = 19, xlab = "Sample size", ylab = "SEM")
lines(x = sample.sizes, y = sigma/sqrt(sample.sizes), col = 'blue')
```



In the last lines of the code shown above we introduced `plot()` function: the first argument is the vector of x-coordinates, the second argument is the vector of corresponding y-coordinates, and the function adds each data point (x_i, y_i) to the plot. In our case, x coordinates are sample sizes N and y coordinates are SEMs

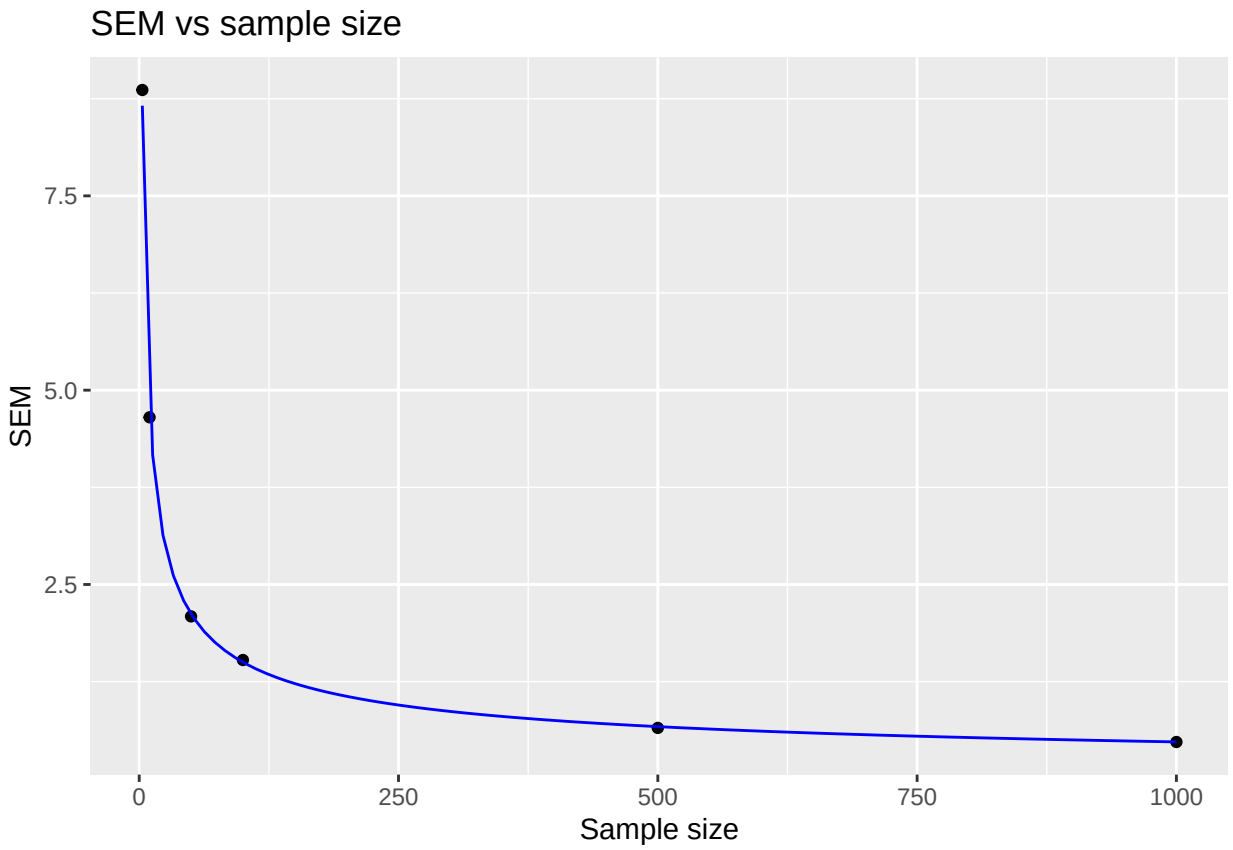
we just calculated. By default, `plot()` draws only data points themselves (without connecting lines, which also can be done). The last command calls the function `lines()` which is in fact a wrapper for the same function `plot()`, but has different defaults that are more convenient to us here: first, it does not start a new plot (which is default behavior of `plot()`), but instead adds to the existing one; second, it draws lines connecting the the data points. The data points we specify for this function are calculated according to the theoretical prediction: it can be shown that when samples of size N are repeatedly drawn from a distribution with standard deviation σ, the standard error of the mean (i.e. the standard deviation of the *distribution of the means* of such samples) is $SEM = \frac{\sigma}{\sqrt{N}}$.

Thus if you play with this code (please do!) and decide to try drawing samples from a distribution with a different standard deviation, do not forget to use correct σ in the last drawing command (in the code above we are using `1/sqrt(sample.sizes)`, i.e. we assume that samples are drawn from the distribution with $\sigma = 1$, just like we did in class when we used standard normal distribution with mean $\mu = 0$ and standard deviation $\sigma = 1$). HINT: your simulated SEM values should fall nicely onto the theoretical curve. If they don't, you got something wrong!

For the full credit on this problem, you have to practice working with R's documentation. Please see the docs (execute `help(plot)` or simply `?plot`) and find out what you need to add to the `plot()` command in the starter code to set the axis labels. Your resulting plot **must** have X-axis labeled as "Sample size" and y axis labeled as "SEM". This last part will cost **5 points** towards the full 30 point credit for this problem.

If you prefer using `ggplot2` as your plotting facility in R (in which case you will know how to use `stat_function` to add theoretical curve to a scatterplot), please feel free to accomplish the above goals using it instead of base graphics plotting functions shown above.

```
ggplot(data = data.frame(sample.sizes, mean.sds), aes(sample.sizes, mean.sds)) + geom_point() + stat_fu
```



Problem 2 (30 points).

There is a beautiful fact in statistics called the Central Limit Theorem (CLT). It states that the distribution of a sum of N independent, identically distributed (i.i.d.) random variables X_i has normal distribution in the limit of large N, regardless of the distribution of the variables X_i (under some very mild conditions, strictly speaking).

Here is what it means in plain English: suppose we have a random variable distributed according to some arbitrary distribution $F(x)$ (i.e. we have a "measurement process" that returns values according to $F(x)$). Let's "measure" the variable, i.e. draw a value from that distribution, x_1. Then let us draw another value x_2 from the same distribution, independently, i.e. without any regard to the value(s) we have drawn previously. Continue until we have drawn N values: $x_1, \ldots, x_N$.

Let us now calculate the sum $s = \sum_1^N x_i = x_1 + \ldots + x_N$ and call *this* an "experiment". Clearly, s is a realization of some random variable: if we repeat the experiment (i.e. draw N random values from the distribution again) we will get a completely new realization $x_1, \ldots, x_N$ and the sum will thus take a new value too! Using our notations, we can also describe the situation outlined above as

$$S = X_1 + X_2 + \ldots + X_N, \quad X_i \text{ i.i.d.}$$

The fact stated by this equation, that random variable S is the "sum of random variables" is just what we discussed above: the "process" S is *defined* as measuring N processes which are "independent and identically distributed" (i.e. draw from the same distribution, without any regard for the values that were already drawn in the past) and summing up the results.

We cannot predict, of course, what the sum is going to be until we perform the actual measurement of $X_1, \ldots, X_N$, so S is indeed a random variable itself! Thus it has some probability distribution/probability density associated with it (some values of this sum are more likely than others), and what CLT tells us is that at large N this distribution is bound to be normal, *regardless* of $F(x)$ we are drawing variables X_i from.

Instead of proving CLT formally, let's simulate and observe it in action.

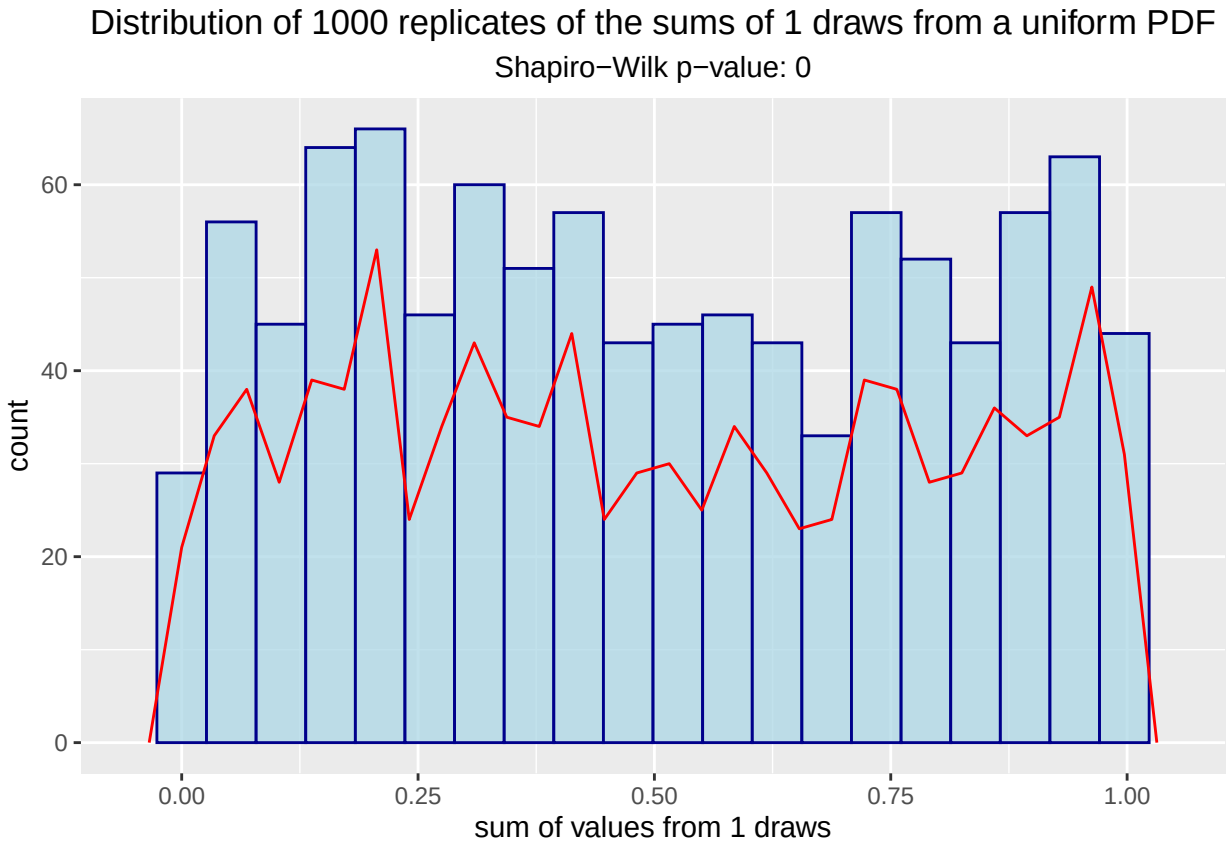
Here is initial code you will have to complete (remember about `eval=FALSE`!):

```
# Set up ggplot to use centered titles
theme_update(plot.title = element_text(hjust = 0.5),
             plot.subtitle = element_text(hjust = 0.5))

clt <- function(n){
  # how many times we are going to repeat the "experiment" (see the text above for what we now call an
  n.repeats <- 1000
  s.values <- numeric() # we will use this vector to store the value of the sum, s, in each experiment

  for (i.exp in 1:n.repeats) { # repeat the experiment 'n.repeats' times
    # More details below. In each "experiment" we must draw the values x1, ..., xN of all the
    # random variables we are going to sum up:

    ### replace with correct call:
    x <- sum(runif(n))

  # the "measured" value of the random variable S in the current experiment is the sum of x1...xN;
  # calculate it and save into the vector s.values:

  ### replace with correct call:
  s.values[i.exp] <- x
  }
```

```r
  # we repeated the experiment 1000 times, so we have 1000 values sampled from the random variable ("pr
  # S and that should be plenty for looking at their distribution:

  ### replace with correct call:

  # Test for normality (just for fun)
  shapiro.p <- round(shapiro.test(s.values)$p.value, digits = 4)

  # PLOT:
  g <- ggplot(data = data.frame(s.values), aes(s.values))
  g <- g + labs(title = paste("Distribution of", n.repeats, "replicates of the sums of", n, "draws from
                x = paste("sum of values from", n, "draws"), subtitle = paste("Shapiro-Wilk p-value:", s
  g <- g + geom_histogram(bins = 20, col = "darkblue", fill = "lightblue", alpha = 0.75)
  g + geom_freqpoly(col = "red")


  }

# Define sequence of n values
N <- 2 ^ c(0:10)

# calculate/plot for each value in N
lapply(N, clt)
```

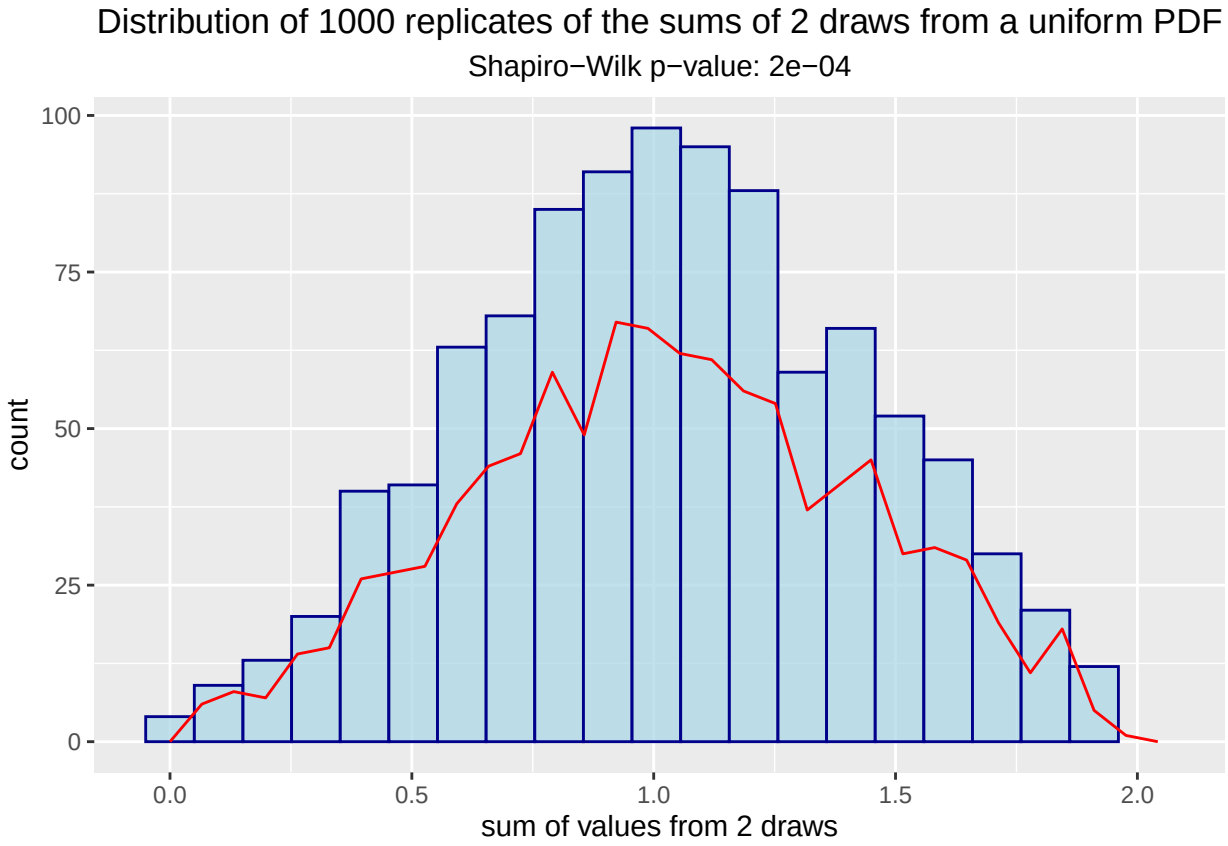
```
## [[1]]
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



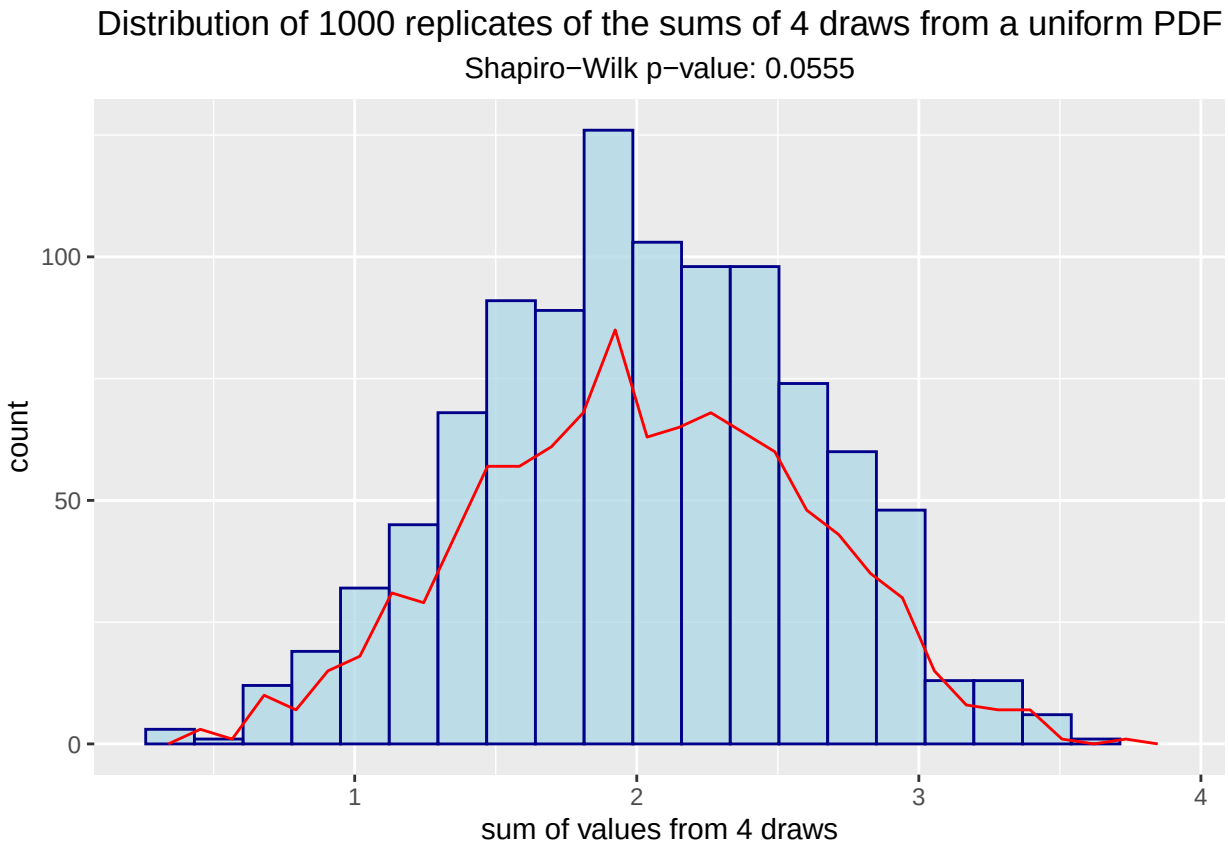
```
##
## [[2]]
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



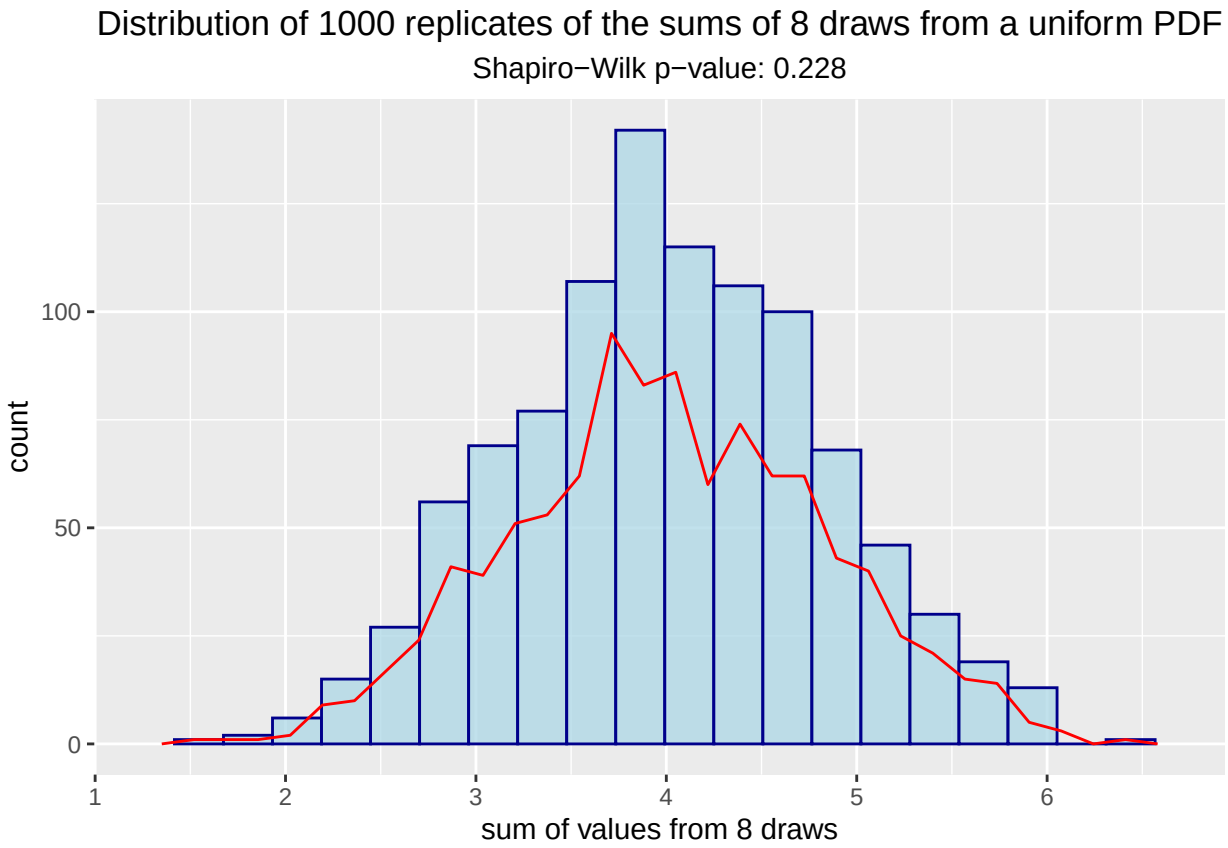

```
##
## [[3]]
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



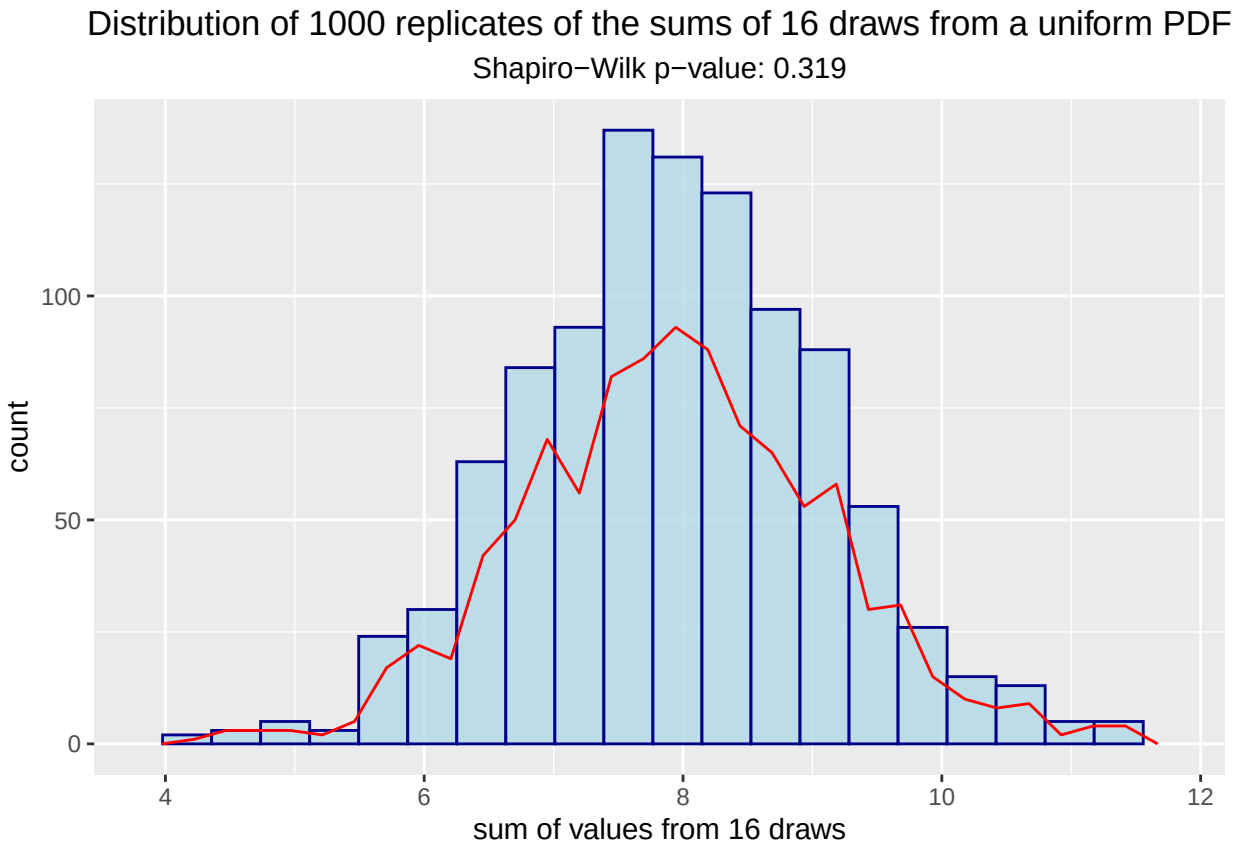

```
##
## [[4]]
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



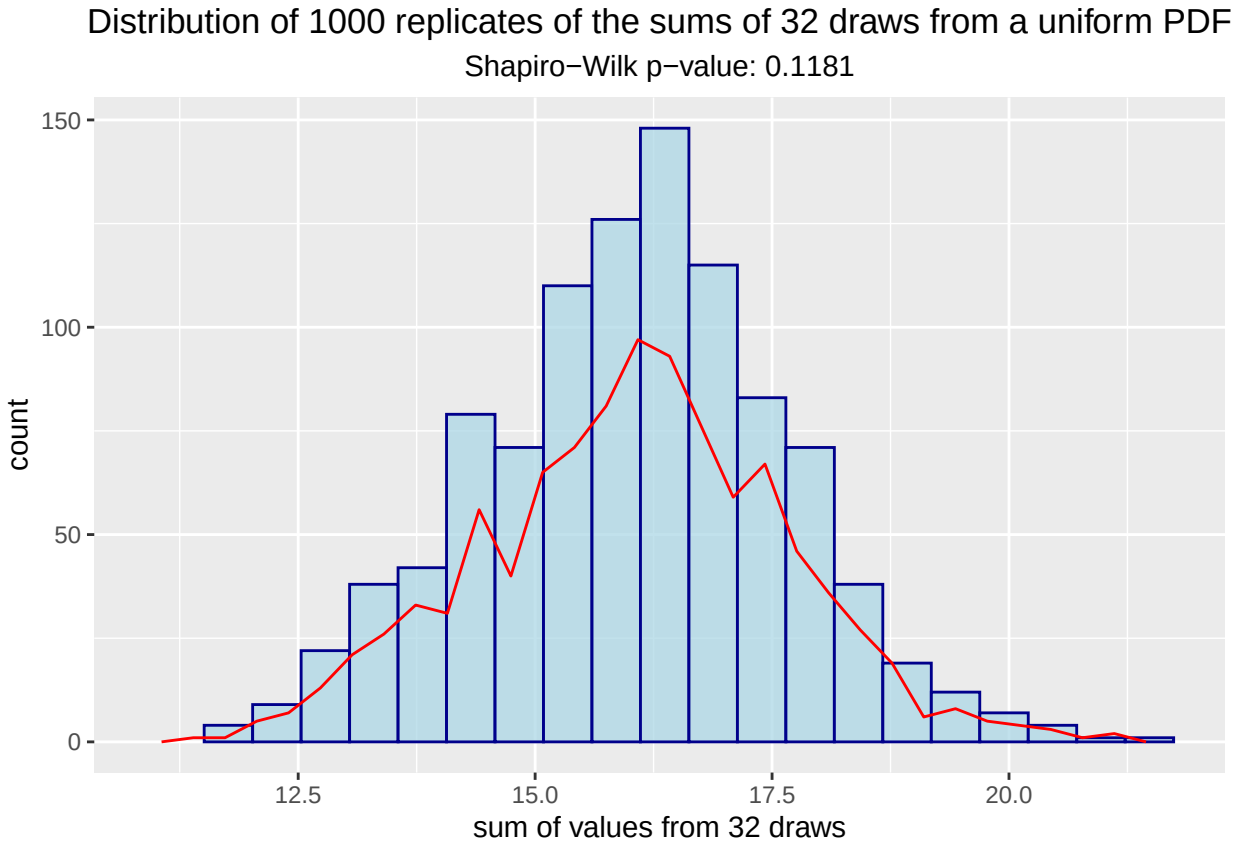

```
##
## [[5]]
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



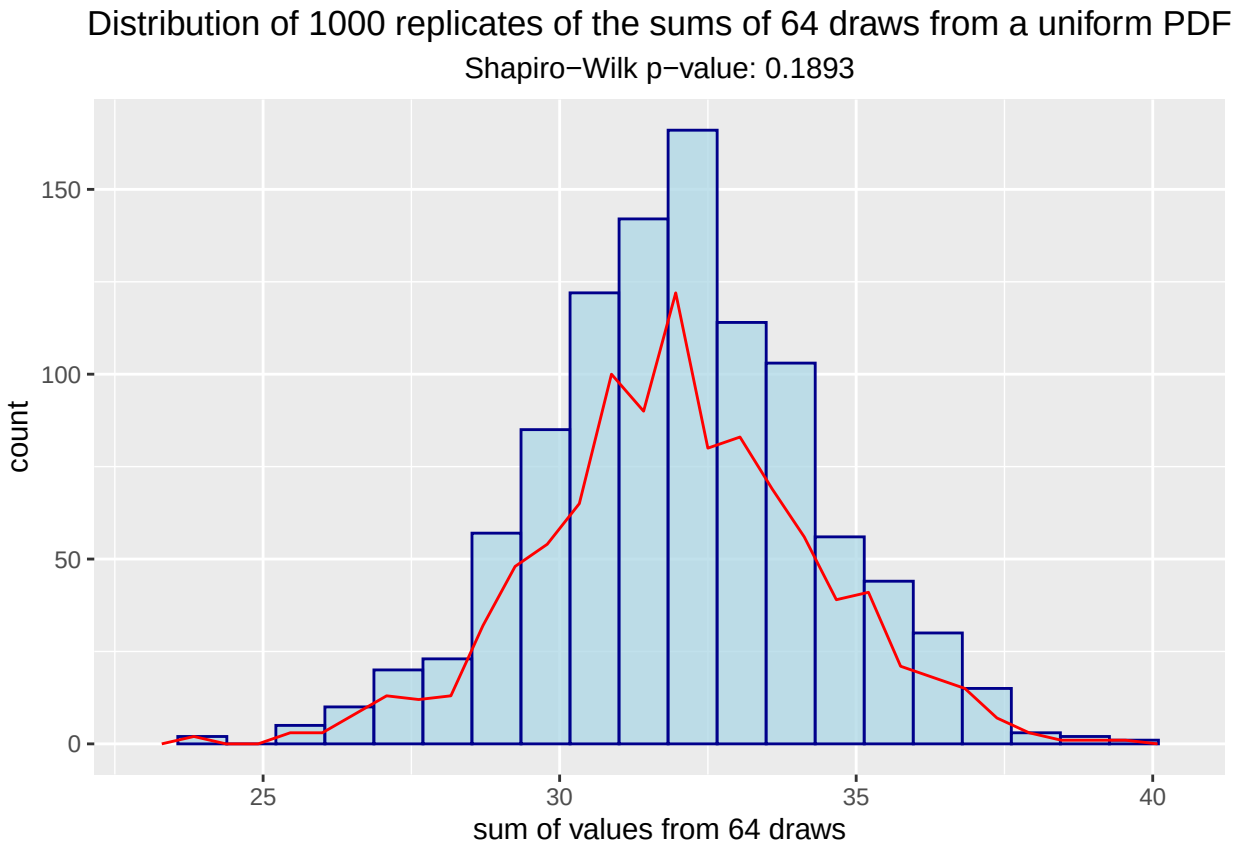

```
##
## [[6]]
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



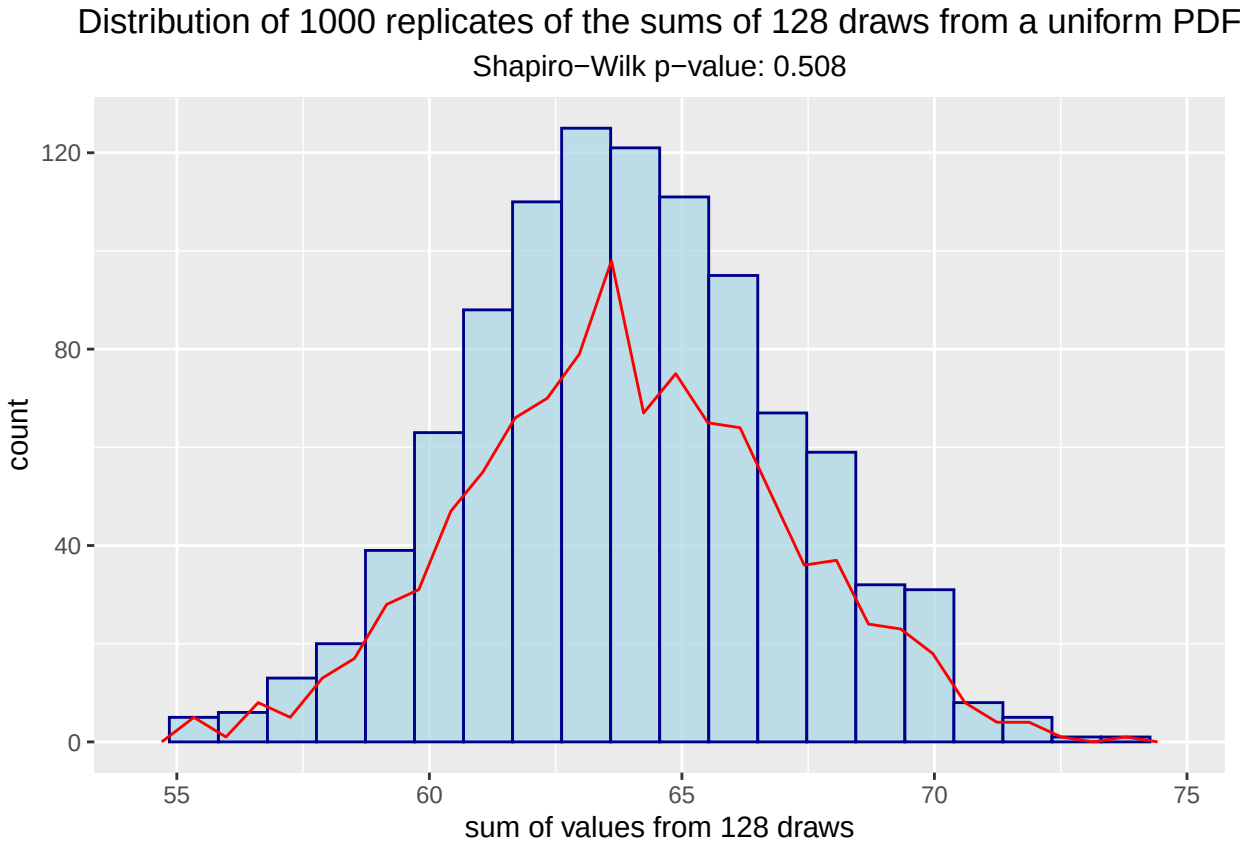

```
##
## [[7]]
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



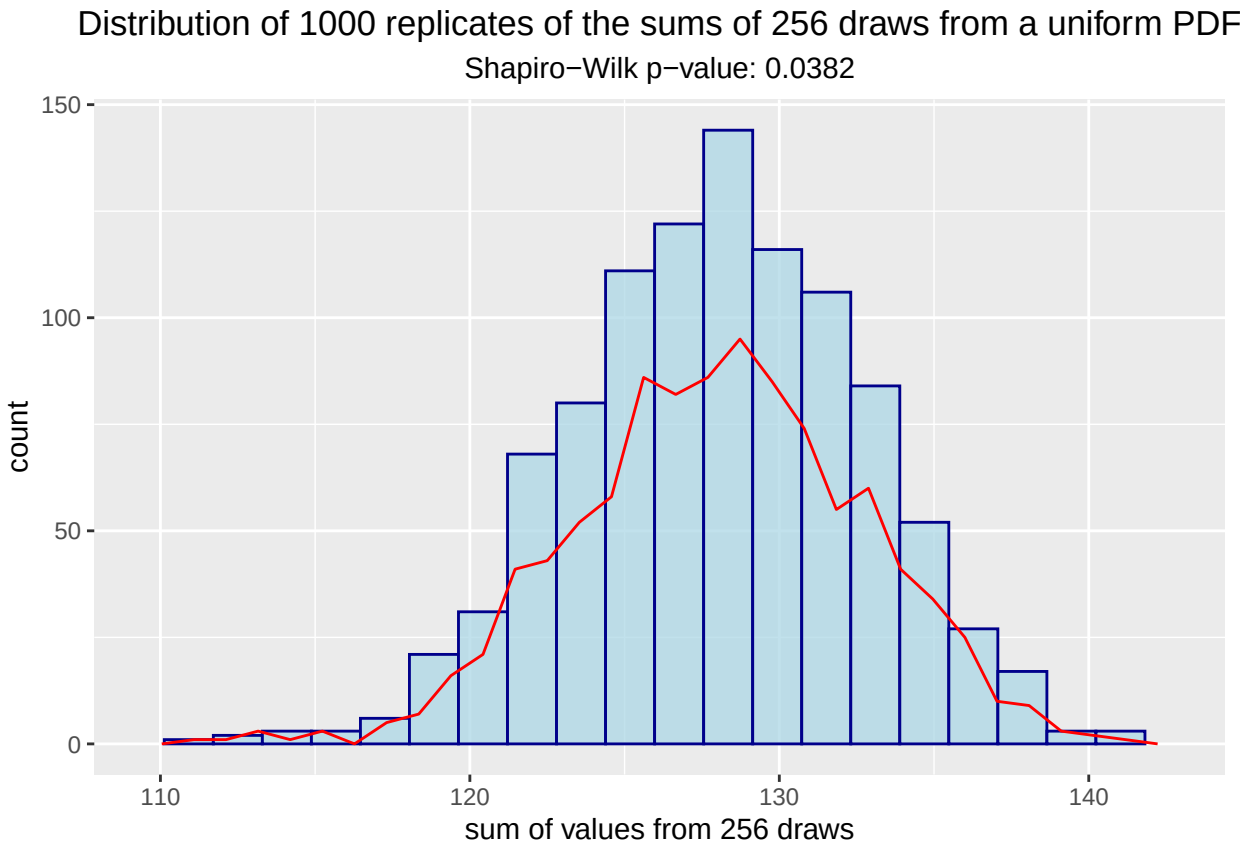

```
##
## [[8]]
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



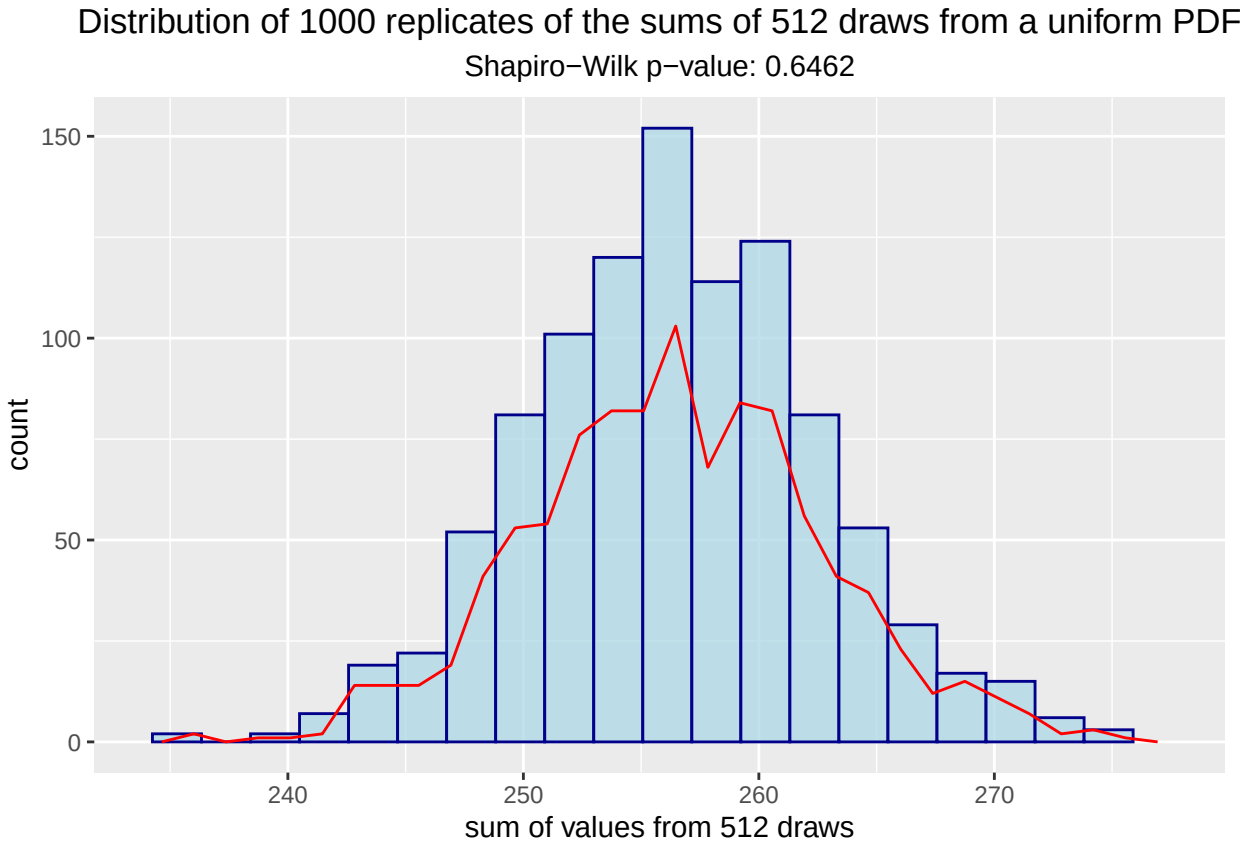

```
##
## [[9]]
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



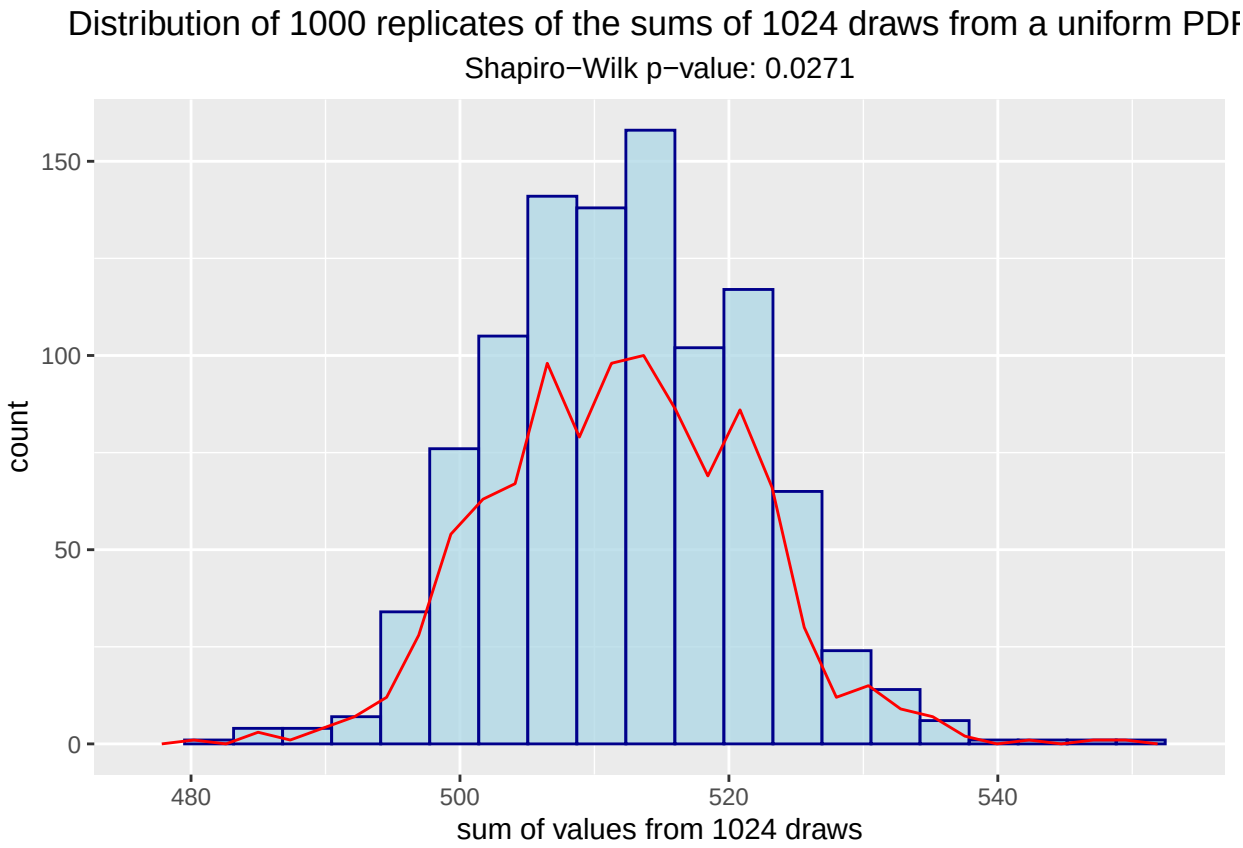

```
##
## [[10]]
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
##
## [[11]]
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



What you need to do is

1. provide missing pieces indicated in the code skeleton above. ✓
2. run the code for few different values of N, *in a loop* (i.e. not by copy-pasting the code manually), see below for the details. ✓(`lapply()` instead of an actual `for` loop)

You should also remember that the sampling functions provided in R do just what we need. For instance, `rnorm(3)` will draw *three* values at once, independently, from the same normal distribution (with default $\mu = 0$ and $\sigma = 1$ in this particular example). But that's exactly what measuring three i.i.d normally distributed random variables is! So in order to sample our N variables $X_1, \ldots, X_N$ in each experiment, we can just call the sampling function once, with N as an argument (and whatever other arguments that specific DISTR function might require).

Please do not use `rnorm()` for this problem though, it is too dull (the sum of *any* number of normally distributed variables, even just two, is again normal!). Use something very different from normal distribution. Uniform distribution or exponential (as implemented in R by`runif()` and `rexp()` functions) are good candidates (see help pages for the distribution function you choose in order to see what parameters it might require, if any). It is also pretty entertaining to see the sum of discrete random variables (e.g. binomial) starting to resemble normal as N increases!

Note that the starter code above uses $N = 1$. In this case $S = X_1$ and obviously S is thus the same "process" as X_1 itself. So the histogram at $N = 1$ will in fact show you the distribution you have chosen for X. Loop over multiple values of N to rerun the code a few times. See how the distribution of S (the histogram we plot) changes for $N = 2$, $N = 5$, ... Can you see how the distribution quickly becomes normal even though the distribution we are drawing values from (the one you have seen at $N = 1$) can be very different from normal?

Your solution for this problem **must include** histogram plots generated at few different N of your choosing, at the very least for (1) $N = 1$ (i.e. the distribution you choose to sample from), (2) N large enough so that the distribution of S in the histogram looks very "normal" , and (3) some intermediate N, such that

distribution of S already visibly departed from $N = 1$ but is not quite normal just yet. The plot titles **must indicate** which distribution and what sample size each of them represents.

Lastly, **for the full credit you should answer the following question (5 points)**: suppose you have an arbitrary distribution and take a sample of N measurements from it. You calculate the mean of your sample. As we discussed, the sample mean is a random variable, of course. How is the sample mean, as a random variable, distributed? What is *its* (expected) mean (zero? infinity? constant? which one if so?)? What about standard deviation of the sample mean? How does it behave as sample size N increases? Can anything be said about the shape of the distribution of sample means in the limit of large N? HINT: look at the definition of the sample mean!