Confidence Intervals

The goal of this lab is to understand and practice testing normality and calculating confidence intervals. We start again by expanding our knowledge of R through the use of for-loops, which you will employ to test data for normality and to compute confidence intervals.

After this lab, you should be able to:

- Test data for assumptions of normality
- Compute and interpret confidence intervals for small and large samples
- Use for-loops to run calculations or produce figures multiple times.

At the end of the lab, there are two problems to solve. Please show your code, type your answers to each problem, and plot graphs using R Markdown. *Submit your answers and R-code to the class Sakai site under the Assignments folder before 5 pm on either Mon. (Section 03) or Wed.(Sections 01 and 02).*

More functions in R

In this lab, we introduce a few new R commands.

- `curve()` - draws a curve corresponding to a function (equation) over a defined interval
- `qqnorm()` - produces a normal QQ plot of the values in a vector
- `qqline()` - adds a theoretical line to the QQ plot based on a normal distribution
- `abline()` - adds a straight line to an existing plot
- `shapiro.test()` - performs the Shapiro-Wilk test of normality
- `runif()` - selects a random variable from a uniform distribution, defined by minimum and maximum values

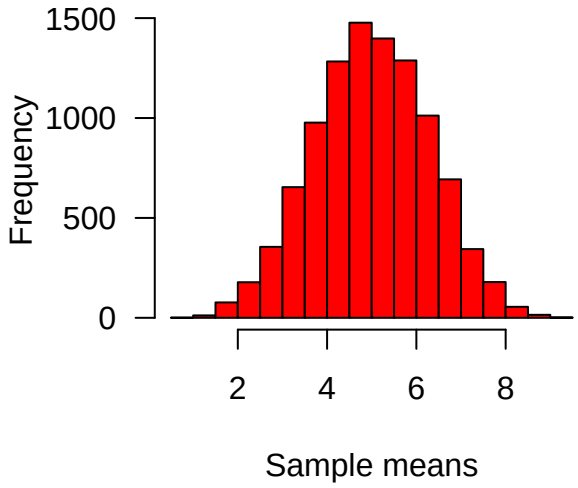
Frequently in statistical analysis, it is important to be able to run calculations or functions multiple times. One intuitive way to do this is by using a for-loop. For-loops are a little bit controversial because running loops is slower than other methods (for example, using the `apply` family of functions). However, for-loops are intuitive and a good place to start before advancing into more efficient commands. The basic set-up looks like this:

```
for (counter in vector) {commands}
```

For example, let's illustrate the Central Limit Theorem (CLT) using a for-loop to pick random numbers from a uniform distribution. The CLT states that if you take repeated samples from a population with finite variance and calculate their averages, then the sampling distribution of the sample means will be normally distributed even if the underyling distribution is not normally distributed.

```
# Create a vector to store results

umeans <- numeric(10000)
set.seed(1001)

# Run the loop 10,000 times

 for (i in 1:10000){

# Take the mean of a sample of 5 uniformly distributed numbers
# and store it in the vector, means
```

```
  umeans[i] <- mean(runif(5, min = 0, max = 10))
 }

# Make a histogram of the 10,000 means
hist(umeans, las = 1, main= "", xlab = "Sample means", col = 2)
```



We will use this example below, but for now you should pay attention to a few things. First, notice that we had to create a vector, `umeans`, to store stuff prior to running the loop. Second, the number of loops is determined by the `for` command. In this example, it runs from 1 to 10,000: i starts at 1 and counts up to 10,000 as the loops are completed. The start and end numbers can be defined by either a number or a function that returns a number (e.g. for `(i in 1:length(umeans)){...}`). Third, note that the new vector has to be subscripted with square brackets, `umeans[i]`, which defines each of the i values of means.

If the above is not crystal clear to you, then re-run the code so that the loop runs from 1 to 10. Make sure you know what each step of the code does.

To Do

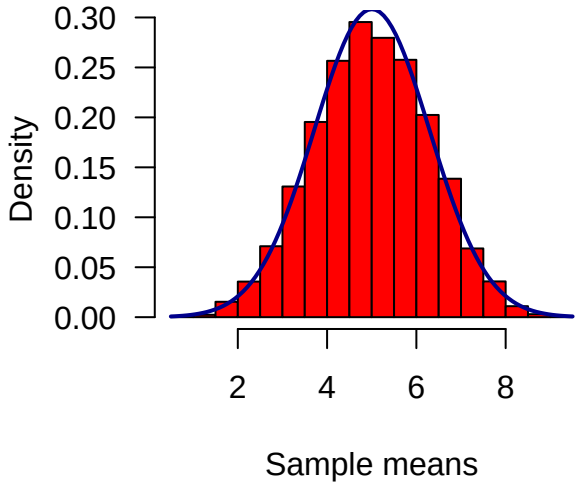
Create a for-loop to take the square root of every element of a vector from 1 to 20.

Testing for Normality

Now let's look at a few ways to assess whether data are normal distributed. Let's start with the data we just created, and then look at a more typical dataset. The histogram of means from above looks close to a normal distribution. But is it?

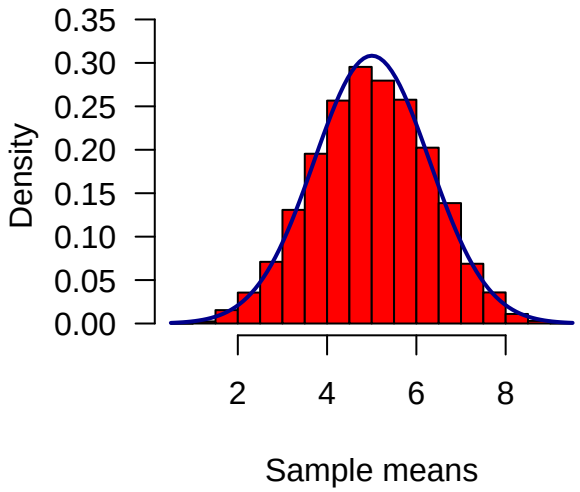
One way to assess normality is to draw a normal distribution with the same parameters on top of the histogram. Note that our previous histogram graphed the "Frequency"" of observations on the y-axis, which ranged from 0 to 1500. If we fit a probability density curve on top of it, the curve will look like a flat line because the probability density function has an integral of 1.0 (the area under the curve). In other words, the scales of the histogram and the pdf would not correspond. Therefore, we need to express the histogram in terms of probability (or alternatively, scale our probability density function to our observations – but that is more complicated). This is done by adding the argument `prob = TRUE`.

```r
hist(umeans, las = 1, main= "", xlab = "Sample means",
     col = 2, prob = TRUE)

curve(dnorm(x, mean = mean(umeans), sd = sd(umeans)),
      add=T, col = "darkblue", lwd = 2)
```



Annoyingly, the y-axis is too short on the histogram and the curve gets cut off. So let's change the y-axis limits to accommodate the curve. If we save the histogram object to a variable, h, then we can see all the "results" from the histogram and use it to modify our y-axis limits. Don't worry about the warning message, R doesn't like that we have suppressed the plotting by creating h.

```r
h <- hist(umeans, prob = T, plot = F)

# Set the range of the y-axis to go from 0 to the maximum density + 15%

hist(umeans, las = 1, main= "", xlab = "Sample means",
     col = 2, prob = T, ylim = c(0, max(h$density)+ max(h$density)*0.15))

curve(dnorm(x, mean = mean(umeans), sd = sd(umeans)),
      add=T, col = "darkblue", lwd = 2)
```

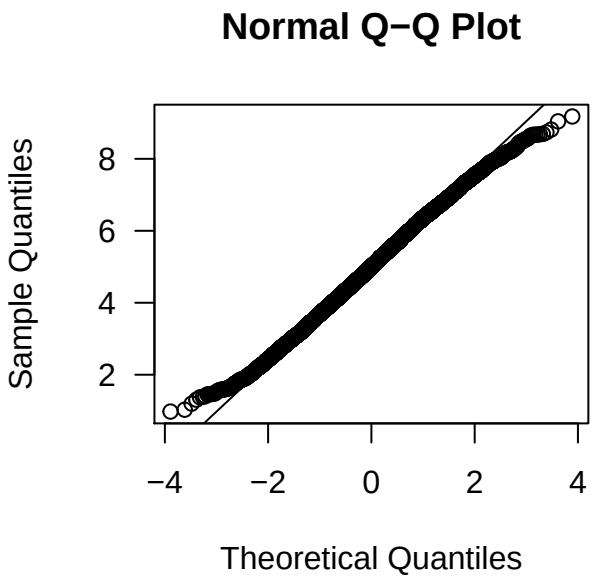


This looks very nice, but is a rather subjective way to evaluate the normality of the data. A more objective

method is to use q-q plots (quantile-quantile plots). The q-q plot is a graphical device used to check the validity of a distributional assumption for a data set. The basic idea is to compute the theoretically expected value for each data point. If the data indeed follow the assumed distribution, then the points on the q-q plot will fall approximately on a straight one-to-one line. The q-q plot provides a visual comparison of the sample quantiles to the corresponding theoretical quantiles.

This is quickly done in R, with `qqnorm` plotting the quantiles of our data and `qqline` fitting a line from a standard normal distribution as comparison.

```
qqnorm(umeans, las=1)
qqline(umeans)
```



What is R doing to compute these lines? Let's dig in deeper using the Africa Plots dataset (`AfrPlots.csv`). As a reminder, the `AfrPlots` database consists of data from 30 1-ha forest plots. The diameters of the trees in the plots were measured once in 2005 (Census 1) and again in 2009 (Census 2). The database includes information on the aboveground biomass of each plot, `ChaveMoist`, the number of trees, `Trees`, the number of dead trees, `Dead`, and the number of new trees from Census 1 to Census 2, `Recruits`.

Start by making a histogram of the dead trees, removing the data from the first census, `CensusNo`, so that you are only using data from the second census period (`CensusNo == 2`). Does it look like these data are normally distributed? Now write your own q-q plot function, `myqqplot`, by completing the following steps. (This is Problem 1 below.)

1. *Sort the data from the lowest to highest.* Hint: `sort(Dead)`
2. *Convert observations to percentiles.* Let n be the number of observations. The lowest observation, denoted as $x(1)$, is the (1/n)th quantile of the data. A quantile times 100 is the percentile, so $x(1)$ is also the (1/n) x 100 percentile of the data. With this convention, however, the largest observation becomes the 100 percentile of the data, which presents a problem as the 100 percentile of a normal distribution is infinity. To avoid this problem, compute `(i - 0.5)/n` to define the ith largest observation, $x(i)$, as the `(i-0.5)/(n)`th. Hint: `p <- (1:n - 0.5)/n`
3. *Calculate the expected percentiles.* The next step is to determine for each observation the corresponding quantile of the standard normal distribution, using `qnorm()`. This value is the "expected" quantile if the data come from a normal distribution. In other words, $x(i)$ should be close to $x'(i)$ if the distribution follows a normal distribution. Hint: `qZ <- qnorm(p, mean = 0, sd = 1)`
4. *Plot oberserved percentiles vs. expected percentiles.* Plot the values (ordered data, z-score) with the measurement scale along the y-axis and the z-scale along the x-axis. This is a normal probability plot – a scatterplot of the data vs. the expected quantiles.

Does your q-q plot look the same as that produced by `qqnorm`?

If you are really ambitious, add a line to the q-q plot like that done by `qqline`. To do so, figure out the slope, m, of the line and, b, the y-intercept, and then use `abline()` to draw the line. The line should pass through the 25% and 75% quantiles.

Understanding Confidence Intervals

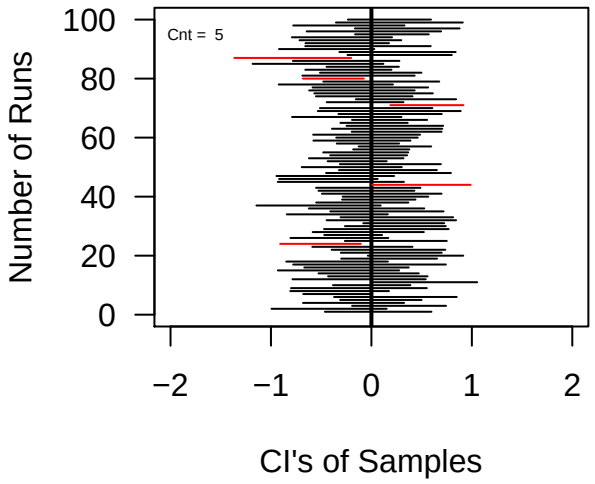
There are two types of estimates for each population parameter: the point estimate and confidence interval (CI) estimate. We first compute the point estimate from a sample. Recall that a sample mean is an unbiased estimate of the corresponding population mean. The confidence interval is a range of likely values for the population parameter based on:

- the point estimate, e.g., the sample mean;
- the desired level of confidence (most commonly 95%, but any level between 0-100% can be selected);
- and the sampling variability, or the standard error of the point estimate.

Strictly speaking a 95% confidence interval means that if we were to take 100 different samples and compute a 95% confidence interval for each sample, then approximately 95 of the 100 confidence intervals will contain the true mean value, μ. Let's see this in action. The below code simulates 100 samples of 15 numbers from a normal distribution with a mean of 0. It then plots the CI's, highlighting those that do not include the true mean.

```
# 1st, set the parameters & variables

mu <- 0
sd <- 1
n <- 15
runs <- 100
ci <- matrix(nrow=runs, ncol=2)

# 2nd, sample from a normal distribution, calculate the mean and
# confidence intervals

 for (i in 1:runs){
     samp <- rnorm(n=n, mu, sd)
          sx <- mean(samp)
     conf <- qnorm(0.975)

     ci[i,] <- c(sx-conf*(sd(samp)/sqrt(n)), sx+conf*(sd(samp)/sqrt(n)))
    }

# 3rd, make an empty graph with a vertical line at 0

par(mfrow=c(1,1))
plot(0,0, xlim=c(-2,2), ylim=c(0,100), type="n",
     xlab="CI's of Samples", ylab="Number of Runs", las=1)
 abline(v=0, lwd=2)

# 4th, add the CI's, coloring them red if they do not overlap the 0 line

cnt <- 0
 for(i in 1:runs){
```

```
      clr <- 1
       if(ci[i,1]>0)clr=2
        if(ci[i,2]<0)clr=2
         if(clr == 2)cnt=cnt+1
      segments(ci[i,1], i, ci[i,2], i, col=clr)
   }

text(-1.75, 95, paste("Cnt = ", cnt), cex=0.5)
```



In practice, however, we select one random sample and generate one confidence interval, which may or may not contain the true mean. The observed interval may over- or underestimate μ. Another way of thinking about a confidence interval is that it is the range of likely values of the parameter (defined as the point estimate + margin of error) with a specified level of confidence (which is similar, but not the same as a probability).

As you see in the code above, a confidence interval for the mean is calculated as the mean $\pm$ the margin of error, where the margin of error is the critical value, z_c, multiplied by the sample standard deviation. For a 95% confidence interval, the area in each tail is equal to $p = \frac{(1-C)}{2} = 0.05/2 = 0.025$. z_c represents the point(s) on the standard normal density curve such that the probability of observing a value greater or equal to z is equal to p. For example, if $p = 0.025$, we need to find the value z_c such that $P(Z \geq z_c) = 0.025$ and $P(Z \leq z_c) = 0.025$. We can do this with:

```
qnorm(p = 0.025, mean = 0, sd = 1)
```

```
## [1] -1.959964
```

Note that this gives you a negative value for z_c. It is easier to work with a positive value, and therefore the statement for the critical values is written to find the upper z_c.

For a population with unknown mean, μ, and unknown standard deviation, σ, a confidence interval for the population mean, based on a simple random sample of size n, is $\overline{x} \pm z_c \cdot \frac{s}{\sqrt{n}}$. In R this can be written as:

```
set.seed(999)
 sample_x <- rnorm(n = 50, mean = 5, sd = 2)
  mean_x <- mean(sample_x)
   z_c <- qnorm(0.975, mean = 0, sd = 1)
    low_x <- mean_x - z_c*sd(sample_x)/sqrt(length(sample_x))
    up_x <- mean_x + z_c*sd(sample_x)/sqrt(length(sample_x))
```

To Do

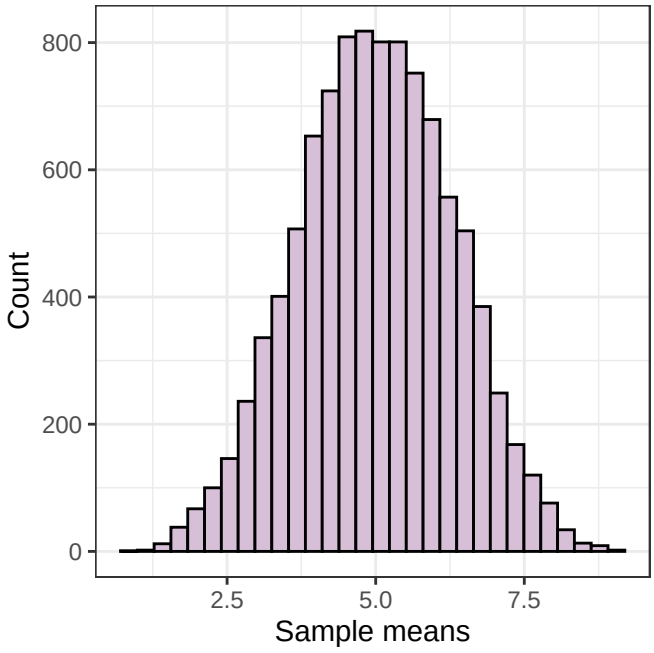
Calculate the 95% confidence interval for the brain measurements in the datafile, `mammals.csv`.

More R Fun

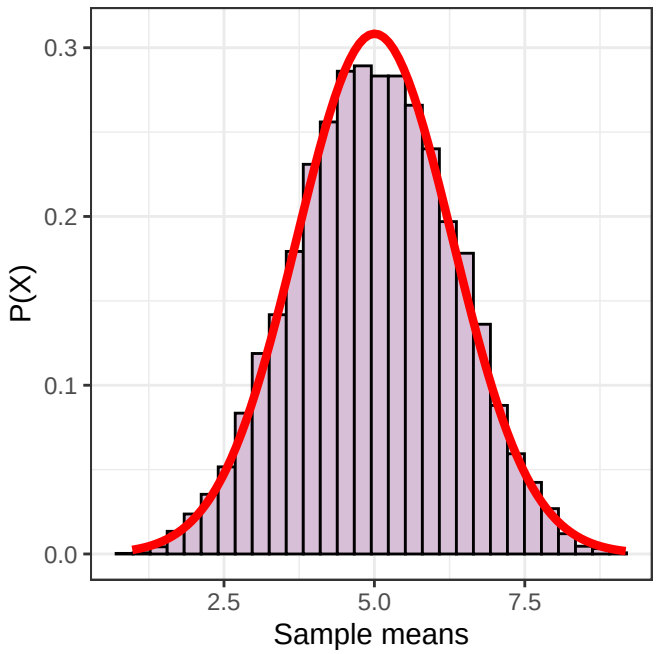
You can ignore this section of the lab, and skip straight to the problems, if you would like. For those students seeking more R, here is a brief intro to the use of `ggplot`. `ggplot2` is a package that makes very attactive and flexible graphs using the *grammar of graphics* that many people prefer to the typical plotting functions. Below I reproduce a few of the graphs above using `ggplot()`. A couple of things to know:

- `ggplot()` doesn't work on vectors, so you need to put the data in a dataframe (see `ums.df` below)
- aesthetic mappings describe how variables in the data are mapped to visual properties (aesthetics) of `geoms`. Aesthetic mappings are almost always set with `aes(...)`
- `qplot()` can be used to make quick plots similar to `hist()`, but to really get the most out of the *grammar of graphics* you need to get into using `ggplot()`
- Try the below plot with and without `theme_bw()` to see what it does
- `stat_function` overlays the normal curve over the histogram using `dnorm()`, with which you should be familiar.

```
# Make the histogram in ggplot
require(ggplot2)
 ums.df <- data.frame(umeans)

  gg1 <- ggplot(ums.df, aes(x = umeans)) +
           geom_histogram(colour = "black", fill = "thistle") +
            xlab("Sample means") + ylab("Count") +
           theme_bw()
   gg1
```



```
gg2 <- ggplot(ums.df, aes(x = umeans)) +
         geom_histogram(aes(y = ..density..), colour = "black", fill = "thistle") +
          stat_function(fun = dnorm,
                        args = list(mean = mean(ums.df$umeans),
                                    sd = sd(ums.df$umeans)),
              lwd = 1.5,
              col = 'red') +
          xlab("Sample means") + ylab("P(X)") +
        theme_bw()
 gg2
```



Problem 1

Turn in the code for your q-q plot function, `myqqplot`. Include the following: (a) a graph of the q-q plot from the `Death` data using `myqqplot`, (b) a graph of the q-q plot from the `Death` data using `qqnorm` and `qqline`. Write a short paragraph describing how q-q plots are created in your own words.

Problem 2

Using the datafile, *epa2012.csv*, evaluate the data for highway gas mileage. Assess whether highway gas mileage is normally distributed, doing the following: (a) plot a histogram of the data with the curve reflecting the expected normal distribution of the data, (b) plot the q-q plot with the one-to-one line, (c) create a boxplot of the data, and (d) calculate the skewness and kurtosis of the data. Do this for both the raw data and log-transformed data. Which looks more normally distributed? Calculate the mean and confidence interval of the data set that you think is mostly likely normally distributed.