ENV 710 – Applied Data Analysis
Fall 2014
Lab 4: Confidence Intervals

The goal of this lab is to gain a better understanding of testing for normality and the use of 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 plots multiple times.

At the end of the lab, there are a few questions to answer. Please type your answers to each problem, including any requested graphs, in a Word document. *Submit your answers and your **R**-code to the class Sakai site under the folder Assignments before 5 pm on either Mon., Sep. 22 (Section 07) or Wed., Sep., 24 (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 provide 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 often slower than other methods (for example, using the `apply` family of functions). However, for-loops are intuitive and a good place to start before we advance into more efficient commands. The basic set-up looks like this:

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

For example, here we will demonstrate the central limit theorem using a for-loop to pick random samples from a uniform distribution. Remember, the central limit theorem states that if you take repeated samples from a population with finite variance and calculate their averages, then the sample means will be normally distributed.

```
means <- numeric(10000)
set.seed(1001)

for (i in 1:10000){

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

hist(means, 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, `means`, 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 (number) and end (number) could be replaced with either a number or a function that returns a number (e.g. `for (i in 1:length(means){…}`). Third, note that the new vector has to be subscripted with square brackets, `means[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.

Testing for Normality

Now let's look at a few ways to assess whether data fit a normal distribution of not. We will 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 test for normality is to draw a normal distribution with the same parameters on top of the histogram. Note that our previous histogram graphed "Frequency" of observations on the y-axis, and we had 10,000 observations. If we try to fit a probability density curve it will look like a flat line because the probability density function has an integral of 1.0 (the area under the curve). 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).

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

curve(dnorm(x, mean = mean(means), sd = sd(means)),
      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.

```
h <- hist(means, prob = T)
hist(means, 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(means), sd = sd(means)),
      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 an graphical device used to check the validity of a distributional assumption for a data set. In general, the basic idea is to compute the theoretically expected value for each data point based on the distribution in question. If the data indeed follow the assumed distribution, then the points on the q-q plot will fall approximately on a straight line. The q-q plot provides a visual comparison of the sample quantiles to the corresponding theoretical quantiles.

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

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

What is **R** doing to compute these lines? Let's dig in deeper and work through it, but 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. Does it look like these data will be normally distributed? Now write your own q-q plot function (`myqqplot`), by completing the following steps.

1. Sort the data from the lowest to highest. Hint: `sort(Dead)`
2. 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 $(1/n) \times 100^{th}$ 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 i^{-th} largest observation, $x_{(i)}$ as the $(i-0.5)/(n)^{th}$. Hint: `q <- (1:n – 0.5)/n`
3. 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 distribution follows a normal distribution. Hint: `qZ <- qnorm(q, mean = 0, sd = 1)`

4. 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 50 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.

```
mu <- 0
sd <- 1
n <- 50
runs <- 100
ci <- matrix(nrow=runs, ncol=2)

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)))
}

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)
```

```
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("Count = ", cnt), cex=0.9)
```

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).

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: Test whether the `ChaveMoist` (for CensusNo = 1) and `Recruits` (CensusNo = 2) data from the African Plots database are normally distributed. For each, show the following: (a) a histogram with normal curve of the data overlain on the histogram, (b) a normal probability plot of the data (e.g. a q-q plot with the data versus the theoretical normal distribution). (c) Also, test each variable for normality using the `shapiro.test()`. Be sure to include the code with which you created these plots.

Explain whether you think these data are normally distributed or not, and why. Describe in words the results of each of the "tests", and for the Shapiro test what the statistic and p-value signify.

Problem #3: Note that the `conf` line in the confidence interval simulation code calculates the statistic associated with a two-sided test and the defined `alpha`. Written above as a Z-statistic, it equals 1.96. Using the confidence interval simulation, change the sample size, `n`, from 50 to 15. How should you change `conf` line given the reduced sample size? Run the simulation a few times (~10) with the appropriate `conf` line and record the approximate proportion of times that the mean is outside of the CI's. Now change `conf` back to `conf <- qnorm(0.975)` and run the simulation the same number of times. Does the proportion of times that the mean is outside of the CI's increase or decrease? Why?

Problem #4: Calculate a 95% confidence interval for the `ChaveMoist` data for Census #2. Write an explanation of what this confidence interval means.