Lab3

```
require(openintro)
```

Confidence Intervals

A confidence interval is an expression of the uncertainty of our estimate. There are two parts: the interval estimate and the associated probability. A 95% confidence interval means if we took repeated samples of the same size from the population and computed our statistic and interval estimate the same way each time, we would *expect* 95% of those intervals to contain the true population parameter.

The 95% confidence interval is generally computed as:[estimate - SE $\cdot$ 1.96, estimate + SE $\cdot$ 1.96]

Note: the 1.96 value changes depending on whether the sampling distribution you refer to is normally distributed or a t distribution, and on what percent you set the confidence interval to be (eg, 95%, 99%, etc.)

Confidence intervals in linear regression

We can compute confidence intervals for the regression coefficients and for conditional expected values of Y ("Y-hat").

```
data("babies")
```

This dataset looks at the birth weights of children and some of the factors that may affect this such as gestation length, mothers' weight, height and age.

```
lm1 <- lm(bwt ~ height, data = babies)
summary(lm1)
```

```
##
## Call:
## lm(formula = bwt ~ height, data = babies)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -65.854 -10.421   0.712  11.446  58.879
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  27.6810    13.0298   2.124   0.0338 *
## height        1.4334     0.2033   7.052 2.97e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 17.94 on 1212 degrees of freedom
##   (22 observations deleted due to missingness)
## Multiple R-squared:  0.03941,    Adjusted R-squared:  0.03862
## F-statistic: 49.72 on 1 and 1212 DF,  p-value: 2.966e-12
```

Now let's find the confidence interval for regression coefficients

```
confint(lm1)
```

```
##                2.5 %    97.5 %
## (Intercept) 2.117489 53.244424
## height      1.034616  1.832255
```

```
27.6810 - 1.96*13.0298
```

```
## [1] 2.142592
```

```
27.6810 + 1.96*13.0298
```

```
## [1] 53.21941
```

```
1.4334 - 1.96*0.2033
```

```
## [1] 1.034932
```

```
1.4334 + 1.96*0.2033
```

```
## [1] 1.831868
```

Find confidence interval for estimated *average* income of people with 13 years of education, also known as E[Y|X=13].

```
predict(lm1, data.frame(height = 60), interval = "confidence", level = 0.95)
```

```
##        fit      lwr      upr
## 1 113.6871 111.7828 115.5913
```

Prediction Intervals

In terms of regression, when we try to predict Y for an individual who has a specific value of X, we still only have our estimated regression line to work with. So our estimate is going to be the same as our estimate for average value of Y at that value of X. The difference is our uncertainty about that estimate. An average value for a group is much easier to estimate than an individual value. Mathematically, you can see the additional uncertainty in the formula for the standard error of a prediction versus a conditional mean. So our interval estimate for the predicted value of an individual will be wider than our interval estimate of an estimated conditional mean.

```
predict(lm1, data.frame(height = 60), interval = "prediction", level = 0.95)
```

```
##        fit      lwr     upr
## 1 113.6871 78.44616 148.928
```

R-squared

You can find the R-squared in the summary of your linear model. The R-squared value tells you the percentage of the variance of Y (outcome) that can be explained X (predictor).

```
summlm1 <- summary(lm1)
summlm1$r.squared
```

```
## [1] 0.03940966
```

Model, Residual and Total Sum of squares

There are a few equations that will be helpful when look at these sum of squares. First is that the total sum of squares is equal to the model sum of squares plus the residual sum of squares. We can extract the residuals using the following code.

```
res <- lm1$residuals
```

It is also helpful to know that:

$$R^2 = \frac{MSS}{TSS} = 1 - \frac{RSS}{TSS}$$

If we have the residuals and the R squared values from the model, we can manipulate these to calculate the other values using algebra.

```
r2 <- summary(lm1)$r.squared

RSS <- sum(res^2)
TSS <- (RSS)/(-(r2-1))
MSS <- TSS - RSS
```

T-tests on the coefficients

The summary output also shows us the result of t-tests done on the intercept and coefficients. These tests useing a null and alternative hypothesis that the coefficient is equal to 0 and not equal to 0 respectively.

```
summary(lm1)
```

```
##
## Call:
## lm(formula = bwt ~ height, data = babies)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -65.854 -10.421   0.712  11.446  58.879
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  27.6810    13.0298   2.124   0.0338 *
## height        1.4334     0.2033   7.052 2.97e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 17.94 on 1212 degrees of freedom
##   (22 observations deleted due to missingness)
## Multiple R-squared:  0.03941,    Adjusted R-squared:  0.03862
## F-statistic: 49.72 on 1 and 1212 DF,  p-value: 2.966e-12
```

If we wanted to do this slightly differently, we might want to look at if the coefficient on mother is actually one. Now this would change both the null and alternate hypothesis to the following:

H0: coefficient = 1 H1: coefficient does not equal 1

We can use the following equation to calculate the t score for this test. We subract 1 as that is our hypothesized value.

$$T = \frac{\hat{\beta}_1 - 1}{SE(\hat{\beta}_1)}$$

```
vcov(lm1)[2,2]^0.5
```

```
## [1] 0.2032799
```

This is the standard error for $\hat{\beta}_1$

To calculate the p-value on for a T statistic you can use the pt function in R.

```
pt(2.124, df=1212, lower.tail=FALSE)*2
```

```
## [1] 0.03387219
```

In a case where the t score is 2.124 and DF=1212 (as in the test for beta0 in the above regression) the p-value is about 0.034. We used lower.tail=FALSE because the t value was positive, and multiplied by 2 to make it a 2-sided test. This means that at the 5% significance level, we can reject the null hypothesis.

#Simulation

We have to do something called "setting the seed" so that our results always come back the same. R generates the random numbers for me, but I have to give it a starting point if I want to run this same simulation later and get the same output.

```
set.seed(12345)
```

First, we set the number of simulations to run. The higher is generally better, but also more computationally intensive for your computer.

```
nsim <- 1000
```

Next, we'll create a population of X and Y values from a normal distribution. The X values will be simulated from a uniform distribution between 0 and 20. The Y values will be simulated such that beta0=5 and beta1=10, but we'll add some error from a Normal(0,2) distribution.

```
beta0 <- 5
beta1 <- 10
N <- 10000
X <- runif(N, 0, 20)
Y <- beta0 + beta1*X + rnorm(N, mean=0, sd=3)
data <- data.frame(X=X, Y=Y)

#Note: the regression for the entire population is very close to what we intended
mod_full <- lm(Y~X, data=data)
summary(mod_full)
```

```
##
## Call:
## lm(formula = Y ~ X, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.0110  -1.9888  -0.0123   1.9911  11.5617
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.955876   0.060333   82.14   <2e-16 ***
## X           10.004417   0.005227 1913.99   <2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.001 on 9998 degrees of freedom
## Multiple R-squared:  0.9973, Adjusted R-squared:  0.9973
## F-statistic: 3.663e+06 on 1 and 9998 DF,  p-value: < 2.2e-16
```

Now, suppose we take a single sample of 100 values from this population and run a linear regression.

```
samp_size <- 100
IDs <- sample(c(1:N), samp_size, replace=FALSE)
samp_data <- data[IDs,]
mod <- lm(Y~X, data=samp_data)
summary(mod)
```

```
##
## Call:
## lm(formula = Y ~ X, data = samp_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.2857 -1.9041  0.0106  1.8103  5.7133
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  5.79809    0.58912   9.842 2.67e-16 ***
## X            9.96116    0.05137 193.928  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.793 on 98 degrees of freedom
## Multiple R-squared:  0.9974, Adjusted R-squared:  0.9974
## F-statistic: 3.761e+04 on 1 and 98 DF,  p-value: < 2.2e-16
```

Now, we'll create a data frame to store our beta1 and beta0.

```
betas <- data.frame(beta0 = rep(NA, nsim),
                    beta1 = rep(NA, nsim))
```

Now, we'll run the simulation where we re-sample each time and save the beta0 and beta1s for that sample

```
samp_size <- 100
for(i in 1:nsim){
  IDs <- sample(c(1:N), samp_size, replace=FALSE)
  samp_data <- data[IDs,]
  mod <- lm(Y~X, data=samp_data)
  betas[i, "beta0"] <- mod$coefficients[1]
  betas[i, "beta1"] <- mod$coefficients[2]
}
```

Now let's inspect the means and standard deviations of the beta0s and beta1s we collected. Note that the means are very similar to the population values from our model using the whole population, and the standard deviations are very similar to the standard errors from our regression run on a single sample of the same size.

```
mean(betas[,"beta0"])
```

```
## [1] 4.976969
```

```r
sd(betas[,"beta0"])
```

```
## [1] 0.5784167
```

```r
mean(betas[,"beta1"])
```

```
## [1] 10.00187
```

```r
sd(betas[,"beta1"])
```

```
## [1] 0.05038256
```

```r
summary(mod) #model based on a single sample
```

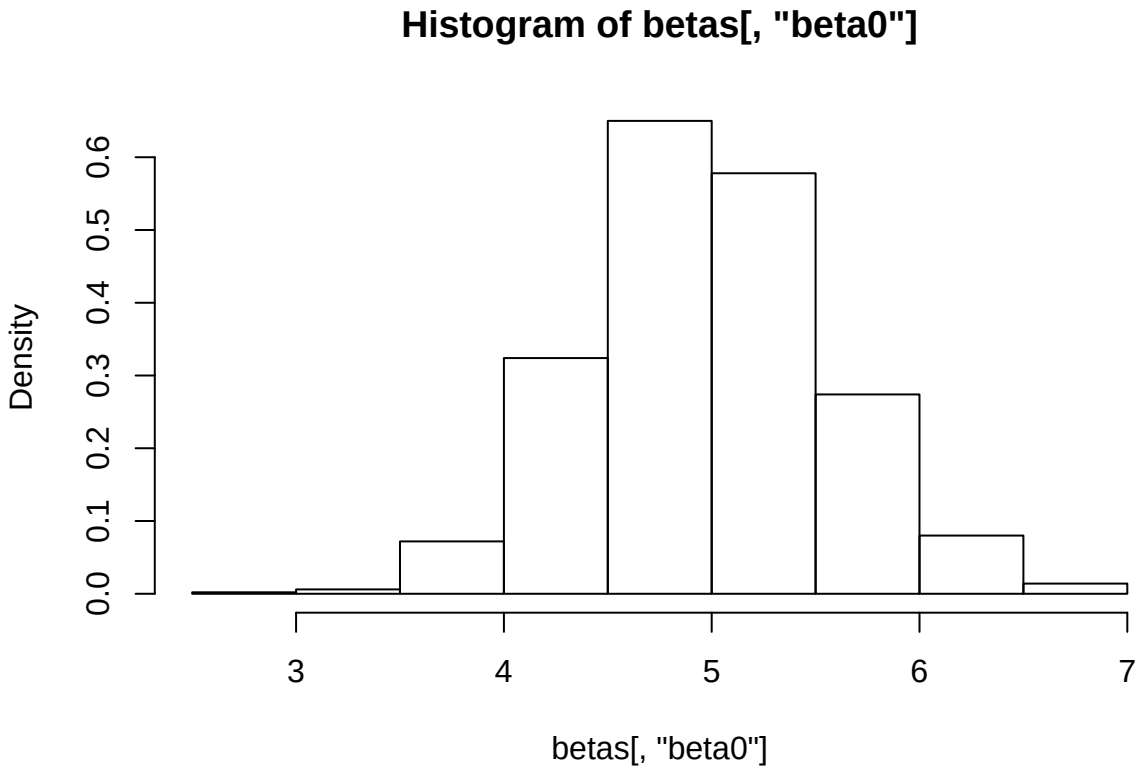
```
##
## Call:
## lm(formula = Y ~ X, data = samp_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.1571 -2.0705 -0.3172  2.1013  8.0496
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.29327    0.60139   7.139 1.66e-10 ***
## X           10.07062    0.05397 186.589  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.06 on 98 degrees of freedom
## Multiple R-squared:  0.9972, Adjusted R-squared:  0.9972
## F-statistic: 3.482e+04 on 1 and 98 DF,  p-value: < 2.2e-16
```

```r
summary(mod_full) #model based on the full population
```

```
##
## Call:
## lm(formula = Y ~ X, data = data)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.0110  -1.9888  -0.0123   1.9911  11.5617
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.955876   0.060333   82.14   <2e-16 ***
## X           10.004417   0.005227 1913.99   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.001 on 9998 degrees of freedom
## Multiple R-squared:  0.9973, Adjusted R-squared:  0.9973
## F-statistic: 3.663e+06 on 1 and 9998 DF,  p-value: < 2.2e-16
```

We can also plot a histogram of the beta0s and beta1s and see that their distribution is essentially normal with means=population values and standard deviations=SEs from the regression with the same sample size

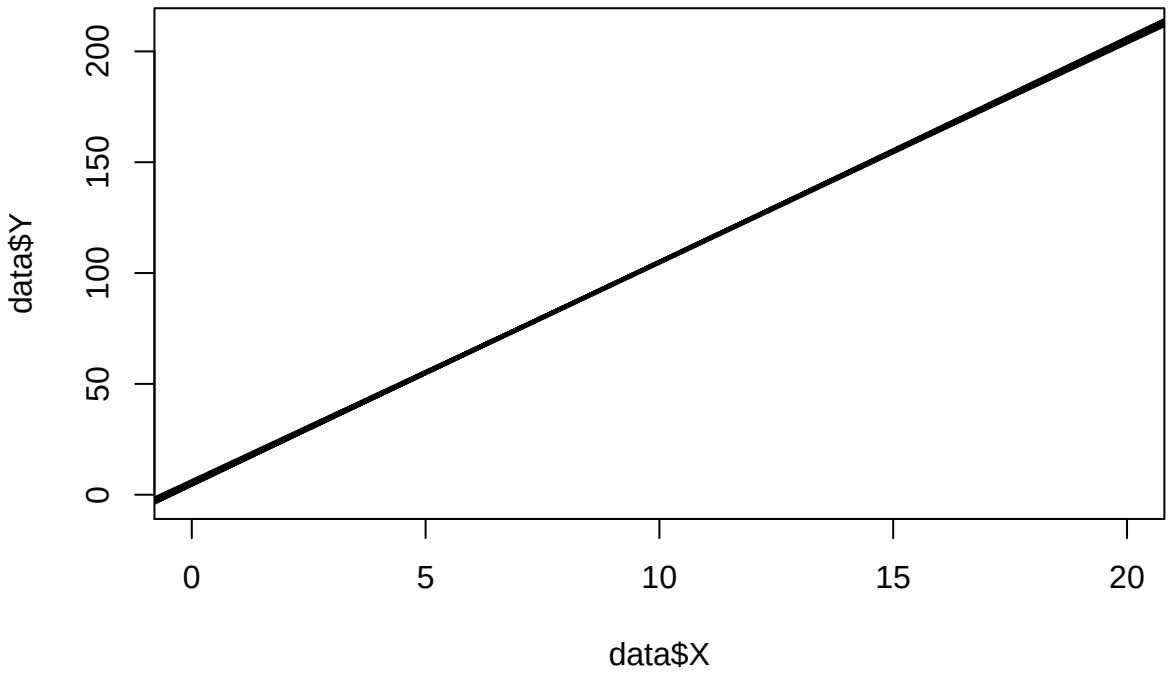
```r
hist(betas[,"beta0"], freq=FALSE)
```




```r
hist(betas[,"beta1"], freq=FALSE)
```

We can also plot the first 100 simulated regression lines.

```
plot(data$X, data$Y, type='n')
for(i in 1:100){
  abline(betas[i,"beta0"], betas[i,"beta1"], col=1)
}
```



Next question: How does the standard error change for different sample sizes? In order to investigate this, let's re-run the simulation with different values of n. We could do this by manually re-changing the sample size and re-running the simulation a bunch of times. Or, we could re-do the simulation with a new parameter!

First, we'll create a vector of some values of sample size, n.

```
n.vec <- c(5, 10, 30, 100, 1000, 3000)
```

Now, we'll create a place to store our results for each value of n:

```
betas <- as.list(n.vec)
for(i in 1:length(n.vec)){
  betas[[i]] <- data.frame(beta0 = rep(NA, nsim),
                           beta1 = rep(NA, nsim))
}
```

Now, let's re-run the simulation for each value of n:

```
nsim <- 100
for(k in 1:length(n.vec)){
  for(i in 1:nsim){
  IDs <- sample(c(1:N), n.vec[k], replace=FALSE)
  samp_data <- data[IDs,]
  mod <- lm(Y~X, data=samp_data)
  betas[[k]][i, "beta0"] <- mod$coefficients[1]
  betas[[k]][i, "beta1"] <- mod$coefficients[2]
  }
}
```

Now, let's make a histogram of the resulting beta0s for each sample size. What do you notice?:

```r
par(mfrow=c(2,3))
hist(betas[[1]][,"beta0"], xlim=c(-3,17),
     breaks=10, main=paste("N=",n.vec[1]), xlab="beta0")
hist(betas[[2]][,"beta0"], xlim=c(-3,17),
     breaks=10, main=paste("N=",n.vec[2]), xlab="beta0")
hist(betas[[3]][,"beta0"], xlim=c(-3,17),
     breaks=10, main=paste("N=",n.vec[3]), xlab="beta0")
hist(betas[[4]][,"beta0"], xlim=c(-3,17),
     breaks=10, main=paste("N=",n.vec[4]), xlab="beta0")
hist(betas[[5]][,"beta0"], xlim=c(-3,17),
     breaks=10, main=paste("N=",n.vec[5]), xlab="beta0")
hist(betas[[6]][,"beta0"], xlim=c(-3,17),
     breaks=10, main=paste("N=",n.vec[6]), xlab="beta0")
```