In this section we'll explore basic hypothesis testing in R. After catching up on a question about looping over variables from last section, we'll work through calculating t- and F-statistics, followed by demonstrating graphically that the t-statistic $t_j = \frac{b_j - \bar{\gamma}}{\text{se}(b_j)}$ is distributed $t_{n-k}$. If time remains,

# Last section

**Looping over variables**: Those of you familiar with Stata know that local macros make looping over variables criminally easy. Unfortunately, we leave that ease behind to some extent when we enter the world of R. The difference is almost philosophical — R is designed to behave more like a traditional programming language in that it has a clear separation between the names of variables and strings. However, there are still a few good ways to loop over variables in R. We'll go through three different methods of drawing multiple histograms in a loop using the variable names.

To begin, we'll load up some variables from the iris dataset built into R:

```
slength <- iris$Sepal.Length
swidth <- iris$Sepal.Width
pwidth <- iris$Petal.Width
```

*Method 1*: The simplest way to deal with these three variables is to load them into a new data frame and to run either an apply function or a loop to produce the histogram. We'll use a for loop:

```
iris.df <- data.frame(slength, swidth, pwidth)
for (i in 1:3) {
  png(paste0("graphs/method1_",names(iris.df)[i],".png"))
  hist(iris.df[, i], main = paste0("Method 1: Histogram of ",names(iris.df)[i]))
  dev.off()
}
```

If we just wanted to draw the histogram, this code could be much simplier — we would just have `hist(iris.df[ ,i])` inside the loop. Instead, though, we use `png()` to draw our histograms in .png files and `dev.off()` to close the .png after we've finished drawing. `paste0()` is a command for concatenating a set of objects into a single string.

*Method 2*: Sometimes, particularly when working with large amounts of data and limited computing resources, we may prefer to use the variables in place, rather than copying them into a data frame. Or, we may have some reason to want to loop over the variable names themselves. The `get()` command can be useful in both of these situations.

```
varlist <- c("slength","swidth","pwidth")
for (var in varlist) {
  png(paste0("graphs/method2_",var,".png"))
  hist(get(var), main = paste("Method 2: Histogram of",var))
  dev.off()
}
```

get() takes in a string and returns the object with that name. The first time the loop runs it executes hist(slength,...), the second time hist(swidth,...), and so on.

*Method 3*: For completeness, I present a third method, using the eval(parse(text = ... )) syntax. This syntax is *very* powerful — it lets you pass a string that contains any R expression and have R evaluate that expression. We won't dwell on this; if for some reason you have a use case where neither of the above two methods work, this kind of solution might let you do what you want. But it won't make you happy.

```
varlist <- c("slength","swidth","pwidth")
for (i in varlist) {
  png(paste0("graphs/method3_",var,".png"))
  evalstring = paste0("hist(",i,",main = \"Method 3: Histogram of ", i,"\")")
  eval(parse(text = evalstring))
  dev.off()
}
```

Yeah. That's not really fun for anyone. Note that I saved evalstring within the loop: this was useful for debugging the string I passed to eval(parse(text = ...)). Let's move on!

## Calculating t-tests and F-tests

First, a basic overview in conducting t- and F-tests. Back to `auto.csv`! At some point I will stop using this data. But not today. We'll start with the usual preliminaries:

```
OLS <- function(y,X) {
  return(solve(t(X) %*% X) %*% t(X) %*% y)
}
data <- read.csv("auto.csv", header=TRUE)
names(data) <- c("price", "mpg", "weight")
y <- matrix(data$price)
X <- cbind(1, data$mpg, data$weight)
```

For reference, consider the regression output from `lm()`:

```
res <- lm(price ~ 1 + mpg + weight, data = data)
coef(summary(res))
summary(res)$fstatistic
```

```
               Estimate   Std. Error    t value    Pr(>|t|)
(Intercept) 1946.068668 3597.0495988  0.5410180 0.590188628
mpg          -49.512221   86.1560389 -0.5746808 0.567323727
weight         1.746559    0.6413538  2.7232382 0.008129813
   value    numdf    dendf
14.73982  2.00000 71.00000
```

Now we'll run OLS and define some useful elements for hypothesis testing using the definitions in lecture notes:

```
n <- nrow(X); k <- ncol(X)
b <- OLS(y,X)
e <- y - X %*% b
s2 <- t(e) %*% e / (n - k)
XpXinv <- solve(t(X) %*% X)
se <- sqrt(s2 * diag(XpXinv))
```

By the way, it's good practice to define intermediate variables like `XpXinv`, `s2`, and `se`. This can be useful for bug-checking and for making your code intuitive. For example, I could have defined `se` as `sqrt((t(y - X %*% b) %*% (y - X %*% b) / (n-k)) * diag(solve(t(X) %*% X)))` (or worse!), which would have been a nightmare to debug or understand.

We can now use the vector of standard errors to calculate our `t` and `p` values for the individual t-tests:

```
(t <- (b - 0) / se)
(p <- apply(t, 1, function(t) {2 * pt(-abs(t), df = (n - k))}))
```

```
           [,1]
[1,]  0.5410180
[2,] -0.5746808
[3,]  2.7232382
[1] 0.590188628 0.567323727 0.008129813
```

Great! We have replicated the `Pr(>|t|)` column of the canned output. Now let's try to replicate the full regression F-statistic. This is a joint test of coefficient significance; are the coefficients jointly different from a zero vector? Max has a great description as to why this is different from three separate tests of significance. For now, note that we are testing joint significance by setting:

$$\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{r} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{1}$$

This is great. This simplifies the equation (2.81), which is fairly daunting at first:

$$F = \frac{(\mathbf{Rb} - \mathbf{r})'[\mathbf{R}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{R}']^{-1}(\mathbf{Rb} - \mathbf{r})/J}{s^2} = \frac{\mathbf{b}'(\mathbf{X}'\mathbf{X})\mathbf{b}/J}{s^2} \tag{2}$$

```
(F <- t(b) %*% (t(X) %*% X) %*% b / (s2*3))
```

```
         [,1]
[1,] 158.1714
```

Uh oh. This is much larger than the reported F-statistic of 14.74. What happened? The problem is that we also included the intercept, whereas `R` assumes that this shouldn't be included in the joint test (why not?). Simplification failed. Let's try again, redefining $\mathbf{R}$ and $\mathbf{r}$ without a restriction on the intercept:

$$\mathbf{R} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{r} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{3}$$

Unfortunately, our formula doesn't simplify as nicely, but we still get to drop the $\mathbf{r}$ vectors.

$$F = \frac{(\mathbf{Rb} - \mathbf{r})'[\mathbf{R}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{R}']^{-1}(\mathbf{Rb} - \mathbf{r})/J}{s^2} = \frac{(\mathbf{Rb})'[\mathbf{R}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{R}']^{-1}(\mathbf{Rb})/J}{s^2} \tag{4}$$

We could simplify a bit more[1], but we've pretty much reached diminishing marginal returns so let's just start calculating:

```
R <- rbind(c(0, 1, 0), c(0, 0, 1)); J <- 2
select.var <- solve(R %*% solve(t(X) %*% X) %*% t(R))
(F <- t(R %*% b) %*% select.var %*% (R %*% b) / (s2 * J))
```

```
         [,1]
[1,] 14.73982
```

It worked! This is, of course, one of the simplest possible F-tests we could conduct, but you can see how it would be easy to construct your own F-tests using this framework.

[1] Note that $\mathbf{Rb} = [0 \ b_1 \ b_2]'$.

## t distribution proof

Max showed in class that the t-statistic $t_j = \frac{b_j - \bar{\gamma}}{\text{se}(b_j)}$ is distributed $t_{n-k}$. We won't go over the proof again, but we will use simulated data to visualize the distributions of $z_j$, $q$, and $t_j$. Part of the purpose of this exercise is to give you practice in simulating data, an immensely valuable tool for testing econometric routines and hypotheses[2]. Our goal is to show graphically that the three test statistics are distributed as follows:

$$z_j \equiv \frac{b_j - \bar{\gamma}}{\sqrt{\sigma^2 \cdot (\mathbf{X}'\mathbf{X})^{-1}_{jj}}} \sim \mathbf{N}(0,1) \tag{5}$$

$$q \equiv \frac{\mathbf{e}'\mathbf{e}}{\sigma^2} \sim \chi_{2(n-k)} \tag{6}$$

$$t_j \equiv \frac{b_j - \bar{\gamma}}{\sqrt{s^2 \cdot (\mathbf{X}'\mathbf{X})^{-1}_{jj}}} = \frac{b_j - \bar{\gamma}}{\text{se}(b_j)} \sim t_{n-k} \tag{7}$$

First, we'll set `reps` (the number of times we'll randomly generate data and test statistics), `n`, and `k`. We'll also create the reps $\times$ $k$ matrices for storing the `z`, `q`, and `t` that we'll create in each loop

```
reps <- 10000; n <- 100; k <- 2
z <- matrix(rep(0,reps*k),ncol=k)
q <- matrix(rep(0,reps),ncol=1)
t <- matrix(rep(0,reps*k),ncol=k)
```

Creating `z`, `q`, and `t` in advance isn't strictly necessary but it's much more efficient to create them now than to have `R` resize them every time we run the loop. Now, the action! Once again we'll using a `for` loop:

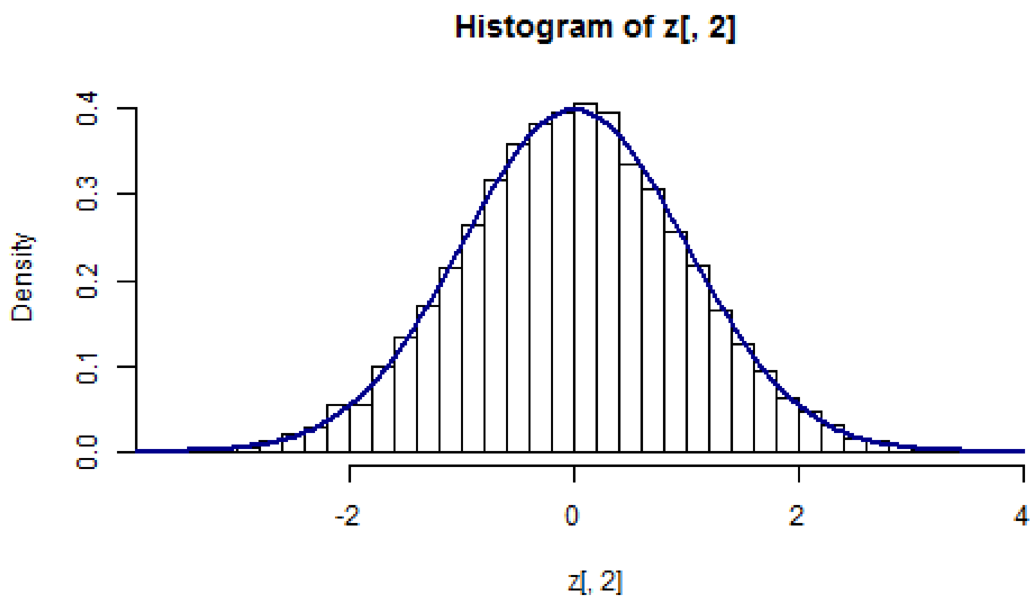
```
for (i in 1:reps) {
  # simulate the true model
  beta <- matrix(c(42,8), nrow=2)
  X <- cbind(1, rnorm(n))
  sigma <- 1
  eps <- matrix(rnorm(n, 0, sigma), nrow=n)
  y <- X %*% beta + eps

  # run OLS and prepare everything we need to calculate z, q, and t
  b <- OLS(y,X)
  e <- y - X %*% b
  XpXinv = solve(t(X) %*% X)
  s2 <- t(e) %*% e / (n-k)
  se <- sqrt(s2 * diag(XpXinv))

  # calculate test statistics
  z[i, ] <- (b - beta) / sqrt(sigma^2 * diag(XpXinv))
  q[i] <- (t(e) %*% e) / sigma^2
  t[i, ] <- (b - beta) / se # t[i, ] <- (b - c(42,7.9)) / se # what if we have the wrong null?
}
```

[2] Note that Max does something similar to construct Figure 2.3 in the notes.

There are three distinct parts to the loop above. First, we simulate a real DGP (including noise), creating `X`, `beta`, and `eps` and then constructing $\mathbf{y} = \mathbf{X}\beta + \varepsilon$. The enormous advantage of simulation is obvious here — since we know exactly what is driving our DGP we can verify that our estimating equation performs as we expect. Next, we run OLS, just as we would if we were presented with a real dataset. Finally, we calculate $z_j$, $q$, and $t_j$.

All that remains now is to compare the simulated distributions of $z_j$, $q$, and $t_j$ to their expected true distributions that we demonstrated in lecture. We'll focus on $z_2$ and $t_2$, since they corresponding to our coefficient $b_2$, which is our randomly generated `X` variable (not the intercept). First, we'll show that $z_2 \sim N(0, 1)$:

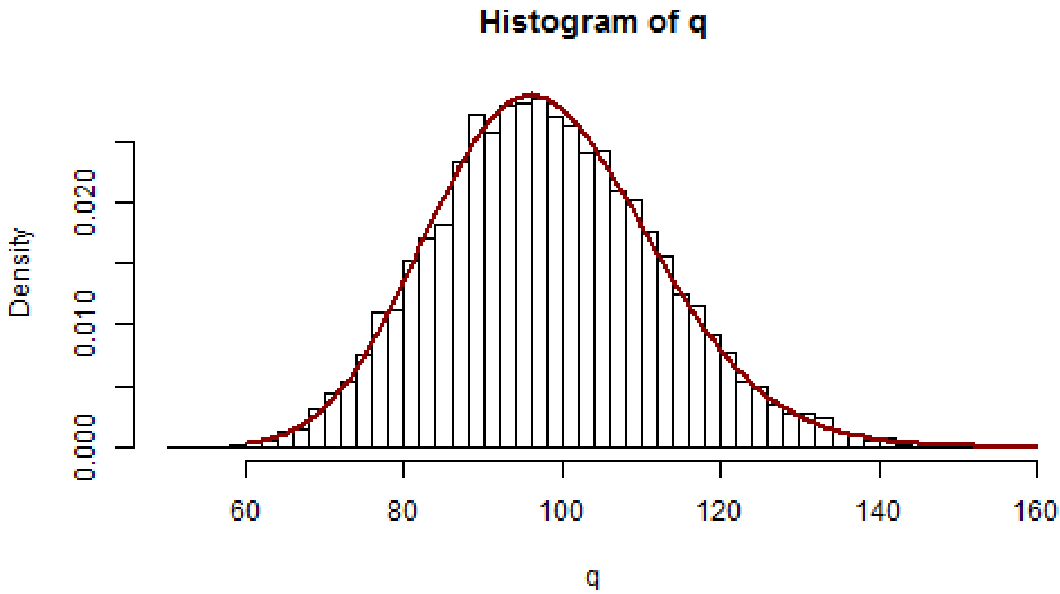
```
hist(z[ , 2], breaks = reps / 200, probability = T)
curve(dnorm(x, mean = 0, sd = 1), from = -4, to = 4, add=T, lwd=2, col="darkblue")
```




There are many prettier ways to plot a graph like this one[3], but this gets the job done. You may notice that we the variable `x` to `dnorm()`, which is a variable we haven't defined. This would normally throw an error, but since `dnorm()` is a function within `curve()`, which accepts functions of `x`, it works as we expect. Now we'll do the same for $q$:

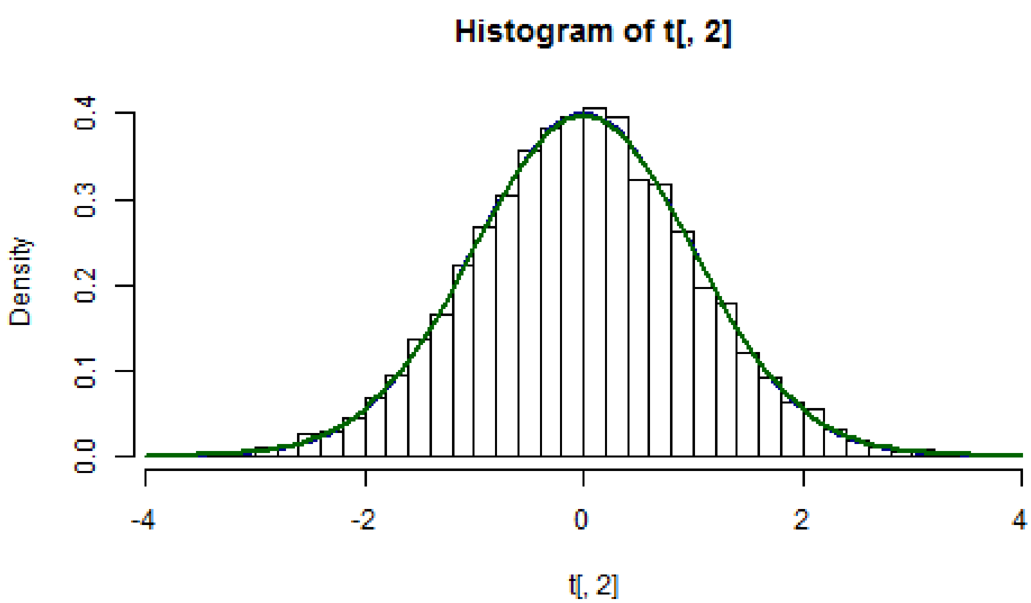
```
hist(q, breaks = 50, probability = T)
curve(dchisq(x, df = n-k), from = 60, to = 160, add=T, lwd=2, col="darkred")
```

[3]For example, using the package `ggplot2`.




We now have graphically demonstrated the truthiness of the two main conditions required to show that $t_2 \sim t_{n-k}$. To complete this exercise, we'll show the result graphically as well.

```
hist(t[ ,2], breaks = reps / 200, probability = T)
curve(dnorm(x, mean = 0, sd = 1), from = -4, to = 4, add=T, lwd=2, col="darkblue")
curve(dt(x, df = n-k), from = -4, to = 4, add=T, lwd=2, col="darkgreen")
```




You'll see in the code that I added a normal curve to the graph on there for good measure. Where is it? It's actually hiding behind the t distribution, since with sufficinetly high degrees of freedom, $\mathrm{df} = \mathrm{n} - \mathrm{k}$, the two distributions are almost exactly the same.

That's it for this section! Next week we'll discuss the first problem set and try our hand at an empirical example that looks at the wage returns to education.

## Puzzle

1. **Partitioned regression**: Generate a $100 \times 4$ matrix $\mathbf{X}$ *including* a column of ones for the intercept. Additionally, generate a vector $\mathbf{y}$ according to the generating process:

$$y_i = 1 + x_{1i} + 2x_{2i} + 3x_{3i} + \epsilon_i,$$

where $\epsilon_i \sim N(0,1)$. Let $\mathbf{Q}$ be the first three columns of $\mathbf{X}$ and let $\mathbf{N}$ be the final column. In addition, let

$$\begin{aligned}
\hat{\gamma}_1 &= (\mathbf{Q}'\mathbf{Q})^{-1}\mathbf{Q}'\mathbf{y} \quad \text{and} \quad \mathbf{f} = \mathbf{y} - \mathbf{Q}\hat{\gamma}_1 \\
\hat{\gamma}_2 &= (\mathbf{Q}'\mathbf{Q})^{-1}\mathbf{Q}'\mathbf{N} \quad \text{and} \quad \mathbf{g} = \mathbf{N} - \mathbf{Q}\hat{\gamma}_2 \\
\hat{\gamma}_3 &= \mathbf{f} \cdot \mathbf{g}/||\mathbf{g}||^2 \quad \text{and} \quad \mathbf{e} = \mathbf{f} - \mathbf{g}\hat{\gamma}_3
\end{aligned}$$

Show that $\hat{\beta} = [(\hat{\gamma}_1 - \hat{\gamma}_2\hat{\gamma}_3) \quad \hat{\gamma}_3]$. Note that the total dimension of $\hat{\beta}$ is 4.