Module 8 Lab

This lab examines permutation tests in R. We use the O-ring example from *The Statistical Sleuth*, section 4.3 (and Module 8, Lecture 1) to walk through running a permutation test with approximate rather than with exact combinatoric methods. For more information on the O-ring dataset, run `?ex2223`; be sure to install and load the *Sleuth3* package first.

The last section also introduces a function for doing the same thing in the *coin* package.

Permutation Test - O-rings

The O-rings, in combination with cold temperatures, were thought to be responsible for the *Challenger* space shuttle explosion. We will look at "O-ring incidents" divided into those occurring below 18 degrees Celsius (cold launches), and those occurring above 18 degrees Celsius (warm launches). (It's important to note that this division is just for pedagogical reasons, allowing us to make a two group comparison. To avoid making such an arbitrary division regression methods should really be used.) The O-ring data is available in the *Sleuth3* package, but we will create it from scratch in the format of slide 9 of Module 8, Lecture 1.

```
cold <- c(1,1,1,3) # number of o-ring incidents in launches below 18 degrees C
warm <- c(rep(0, 17), 1, 1, 2) # ditto above 18 degrees C
```

Notice in defining "`warm`" we use the function `rep(x, times)` which repeats its first argument, `x`, `times` times. I.e. here we repeat zero 17 times, which is less cumbersome than writing out 17 zeroes!

Now we have our data. Our null hypothesis, as in lecture, is that both groups come from the same distribution. Here, instead of using the difference in group means as the test statistic, let's use the t-statistic.

$$t = \frac{\bar{x}_1 - \bar{y}_2}{\sqrt{s_1^2/n_1 + s_2^2/n_2}}$$

This is just a convenient number that summarizes the evidence against the two groups coming from populations with the same means. Because the data is so skewed, we have very different and small samples sizes, and the response is discrete, we don't really think this t-statistic has a t-distribution, that's why we'll compare it to our permutation distribution, not to a t-distribution.

For our data, we can calculate this as follows.

```
T <- (mean(cold) - mean(warm))/sqrt(var(cold)/4 + var(warm)/20) # t-statistic
T
```

```
## [1] 2.531636
```

Or alternatively, and to avoid any arithmetic errors on our part, we could use the `t.test()` function and just pull out the statistic:

```
t.test(cold, warm)$statistic
```

```
##        t
## 2.531636
```

This is our **observed statistic**. To calculate the permutation distribution, which we will compare our statistic to, we need to repeat this calculation on permutations of the group labels.

To randomly permute the groups, we will use the `sample()` function. Let's see how it works before we write a permutation function. The implementation below samples four elements from the integers 1 to 24, without replacement. This means we cannot sample the same number twice.

```
sample(1:24, size = 4, replace = FALSE)
```

```
## [1] 12  5  8 23
```

We'll use this to randomly select four indexes of the original data to serve as the *cool* group. For example, with the above numbers, 12th, 5th, 8th and 23rd observations should be the *cool* group. We had a total of 24 observations, so we need to have some way to number them. Let's just put them together in a vector

```
o_ring_data <- c(cold, warm)
o_ring_data
```

```
##  [1] 1 1 1 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2
```

Then we can take a look at this particular *cool* group, by using indexing

```
o_ring_data[c(12, 5, 8, 23)]
```

```
## [1] 0 0 0 1
```

Or the corresponding *warm* group, by excluding those indexes

```
o_ring_data[-c(12, 5, 8, 23)]
```

```
##  [1] 1 1 1 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 2
```

We then want to use those two groups to calculate our test-statistic. Putting all those steps together, to find one value for our permutation distribution looks like:

```
cold_indices <- sample(1:24, size = 4, replace = FALSE)
cold_perm <- o_ring_data[cold_indices]
warm_perm <- o_ring_data[-cold_indices]
t.test(cold_perm, warm_perm)$statistic
```

```
##         t
## 0.2923308
```

Ignoring the warm/cold labels and pulling four random observations to be the *cold* group, assumes those labels are unrelated to the response, i.e. assumes the null hypothesis is true. So, this t-statistic is one of the equally likely outcomes assuming the null hypothesis is true.

But what we really want to do, is repeat this many times, each time getting one equally likely value of the test statistic under the assumption that the null hypothesis is true. This collection becomes our null distribution.

To do this operation many times, let's turn our "do it once" into a function:

```
perm_o_ring <- function(){
  cold_indices <- sample(1:24, size = 4, replace = FALSE)
  cold_perm <- o_ring_data[cold_indices]
```

```r
  warm_perm <- o_ring_data[-cold_indices]
  t.test(cold_perm, warm_perm)$statistic
}
perm_o_ring()
```

```
##         t
## 0.1895173
```

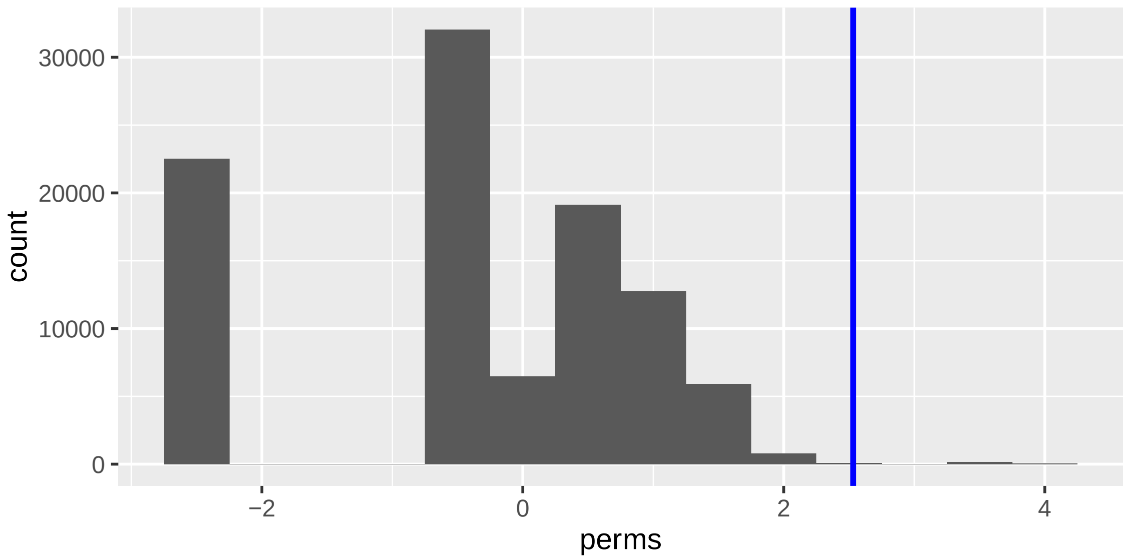
(This is a written a little lazily, see the end of the lab for the problems, but I did it this way so it was easy for you to see how it was turned into a function).

Now we are ready to conduct a permutation test. We want to randomly permute the groups and calculate a test statistic many times to simulate the null distribution. Then we can see how extreme the observed test statistic is compared to that null distribution.

```r
set.seed(1986) # Challenger exploded in 1986
perms <- replicate(100000, perm_o_ring()) # randomly permute, calculate t-stat 100,000 times
mean(perms > T) # Proportion of t-stats more extreme than observed
```

```
## [1] 0.00214
```

```r
library(ggplot2)
qplot(perms, binwidth = .5) + # Histogram of randomly assigned group t-stats
  geom_vline(xintercept = T, color = "blue", size = 1) # vertical line = observed t-stat
```



The first line uses our function `perm_o_ring()` to randomly reassign the groups and calculate the test statistic—100,000 times. This gives the simulated null distribution. The value returned from 0.0021 gives the proportion of outcomes with a test statistic more extreme than the one we observed; this is our permutation test p-value. You can see in the histogram that very few observations lie to the right of the blue line denoting the observed t-statistic.

O-ring Permutation Test - Technical Summary

Based on these 24 launches, there is very strong evidence of a relationship between the number of O-ring incidents and whether the temperatures was below or above 18 degrees Celsius (one-sided permutation test, using a t-statistic, p-value = 0.0021).

O-ring Permutation Test - Coin Package

It's also possible to run a permutation test with a pre-written function in the `coin` package. We can use the same data from before, but we also need to provide group labels,

```
o_ring_data
```

```
##  [1] 1 1 1 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 2
```

```
groups <- factor(c(rep("cold",4), rep("warm", 20))) # Create group label vector
str(groups)
```

```
##  Factor w/ 2 levels "cold","warm": 1 1 1 1 2 2 2 2 2 2 ...
```

The first line returns the data, as a reminder that all observations are included in `o_ring_data`; the first four are "below 18 degrees" observations, the remaining 20 "above 18 degrees" observations. The function we are going to use in the `coin` package, called `oneway_test()`, requires group labels for the data. Therefore, we define `groups` as a factor, which makes sure these are treated as categories, not just at character strings.

We are ready to run the test.

```
library(coin) # load coin package
oneway_test(o_ring_data ~ groups, distribution = "exact")
```

```
##
##  Exact Two-Sample Fisher-Pitman Permutation Test
##
## data:  o_ring_data by groups (cold, warm)
## Z = 3.0604, p-value = 0.009881
## alternative hypothesis: true mu is not equal to 0
```

The output gives the name of the test, and the data used. The third line provides the observed test statistic, albeit a slightly different statistic than before. The third line also provides the p-value, but now it is a two-sided p-value. The final line gives the alternative hypothesis. Notice that because we use `distribution = "exact"` we get the same result as the combinatoric calculation from Lecture 1, Module 8. Use `?oneway_test` to learn more about the function.

For another option, see `permTS()` in the `perm` package, which also has an argument for running an exact permutation test.

Lazy function writing

It's is bad practice for a function to depend on things beyond its inputs. Our `perm_o_ring()` function didn't take **any** arguments, but it relied on us having data already defined in `o_ring_data` and implicitly that this data had 24 observations and the first group had four observations.

A better written function would move these dependencies to arguments, for example, let's make the data and sample size of the first group arguments:

```
perm_o_ring <- function(x, n1){
  n <- length(x)
  cold_indices <- sample(1:n, size = n1, replace = FALSE)
  cold_perm <- x[cold_indices]
  warm_perm <- x[-cold_indices]
  t.test(cold_perm, warm_perm)$statistic
}
perm_o_ring(o_ring_data, 4)
```

```
##         t
## 0.7891376
```

You might then argue that this could be used for any data where you want to use the t-statistic, so we might rewrite the body (but keep the exact same functionality), and give the function a more general name:

```
perm_tstat <- function(x, n1){
  n <- length(x)
  grp1_indices <- sample(1:n, size = n1, replace = FALSE)
  grp1_perm <- x[cold_indices]
  grp2_perm <- x[-cold_indices]
  t.test(grp1_perm, grp2_perm)$statistic
}
perm_tstat(o_ring_data, 4)
```

```
##         t
## 0.2923308
```

You could take this further and let the type of statistic be an argument too, but we'll leave it here for now.