Non-parametric and permutation tests

XDASI Fall 2021

11/4/2021

Contents

So far we have used t-tests to compare two samples, and we've explored transforming our data to make it look more normal so that we can proceed with standard parametric tests.

Two other important approaches are also available that do not assume the data are normally distributed: ***non-parametric*** tests and ***distribution-free*** tests, which look at empirical distributions using ***resampling methods***.

Now that computing power is plentiful and cheap, resampling methods are becoming more common in biology, but standard statistical tests are still the go-to methods for many comparisons.

Non-parametric tests

Suppose we don't want to transform our data and can't assume that the data are normally distributed.

- Instead of using the ***magnitude*** of differences between groups directly, nonparametric tests that look at the data in terms of their ***ranks*** (or ranked differences).
- Rank tests compare ***medians*** instead of means between samples because they treat continuous data as ***ordinal*** rather than ***quantitative*** (interval) data.

The **sign test** and the **Wilcoxon sign rank test** can both be used instead of one-sample or paired t-tests when the data do not meet the assumptions of a t-test (i.e. the data are normally distributed).

Null hypothesis

The intuition behind the null hypothesis for all of these tests is that the ***median difference*** between two samples (or between a sample and an expected value) should be zero, if they belong to the same parent population.

- This means that there should be no ***bias*** in the ***direction*** (sign) of the differences between two groups (or for one-sample tests, between a sample and the expected population mean).
- In addition, if we were to ***rank*** the differences between groups, or just the raw values in each group (depending on the test), they should somehow average out between the two groups. This works well when the two groups have a similar skew and variance.

Test statistics for non-parametric tests

Different tests use different methods to generate a ***test statistic*** that is used to obtain a p-value for observed differences. All of them compute a statistic for both positive and negative differences, and then one of these is chosen to compute the p-value.

Since the amount of bias will be equally extreme in both directions, it doesn't matter which statistic is chosen, but it is important to choose the correct tail of an expected distribution for comparison.

- By default, the statistic that can be compared to a **lower-tail** of a distribution is chosen as the test statistic for obtaining a p-value.

Example: Plasma Triglyceride levels

Last time we used log-transformed values of this dataset to perform t-tests. Here, we will use different non-parametric tests to compare the pre- and post- data.

```
#########################################################################
# plasma triglyceride levels in the population (mg/ml)
# borderline high = 2-4 vs. normal < 2
# testing before and after diet and exercise changes (expect a decrease)
pre = c(2.55,3.38,2.37,4.11,3.27,2.58,4.20,3.22,5.10,2.62,3.06,1.23,2.27,2.24,1.39,2.63,2.61,4.30,1.46,3
post = c(1.59,3.51,1.44,2.32,1.75,1.67,1.90,1.37,2.72,1.80,2.40,2.01,2.41,1.38,1.18,4.31,2.09,2.32,2.63
#########################################################################
```

Nonparametric tests for paired data

Sign test

The sign test is the simplest nonparametric one-sample or paired two-sample test. The hypotheses to be tested are:

H_o: The median paired difference between the groups is zero.

H_A: The median paired difference between the groups is NOT zero.

The sign test simply takes the signs of the paired differences and counts up the number of negative and positive deviations from zero. This generates binary data, so the ***p-value is equivalent to the binomial probability*** for the observed number of negative deviations with an expected probability of $p = 0.5$.

The steps are:

- Calculate the pairwise differences between the samples.
- Count the number of nonzero differences n.
- The test statistic, let's call it k, is the ***smaller*** number of the positive and negative differences.
- Calculate the binomial probability of observing k out of n differences given an expected probability of $\pi = 0.5$.

First, let's compute the test statistic manually. This will be the smaller of the negative and positive counts.

```
# difference in the two samples
tri_diff = post - pre

# count of negative differences
neg_count = sum(tri_diff < 0)
neg_count
## [1] 18

# count of positive differences
pos_count = sum(tri_diff > 0)
pos_count
## [1] 6
```

```
# test statistic
# will be used to get p(X <= x) using lower tail of expected distribution
test_stat = min(neg_count, pos_count)
test_stat
```

```
## [1] 6
```

Since the sign test is just a ***binomial exact test*** using the positive and negative counts, the probability can be computed either using `pbinom()` or using the binomial proportions test, `binom.test()`.

Now we use the test statistic to obtain a two-tailed p-value for the observed data under H_o. This is the two-tailed binomial probability of seeing a value as extreme as our test statistic (here, `neg_count`).

```
# since pbinom(..., lower.tail=TRUE) gives p(X <= x),
# for a 2-tailed test we multiply the probability by 2
2 * pbinom(test_stat, length(tri_diff), prob=0.5)
```

```
## [1] 0.02265584
```

```
# binomial test for counts (default is 2-sided)
binom.test(x=test_stat, n=length(tri_diff), p=0.5)
```

```
##
##
##
## data:  test_stat out of 24L
## number of successes = 6, number of trials = 24, p-value = 0.02266
## alternative hypothesis: true probability of success is not equal to 0.5
## 95 percent confidence interval:
##  0.09773041 0.46711280
## sample estimates:
## probability of success
##                   0.25
```

Wilcoxon signed rank test

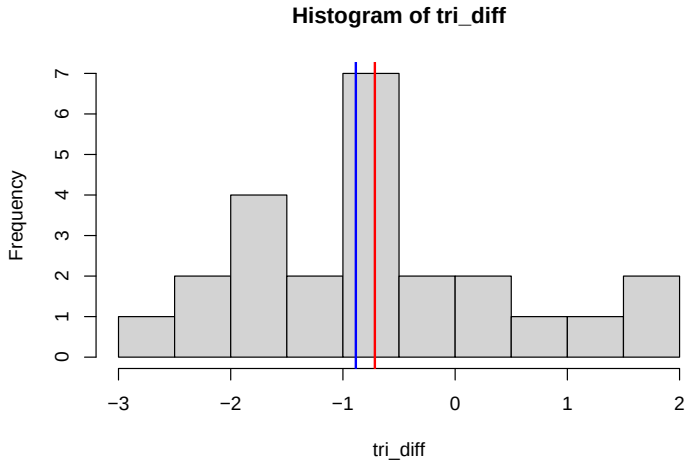
The Wilcoxon signed rank test is similar to the sign test except that it takes the ***sum of all the signed ranks***, and it can easily be performed directly in R.

This test is a non-parametric equivalent to a paired t-test. When the underlying data are normally distributed, it has slightly less ***power*** to detect true differences between groups, but otherwise the opposite is true. We will discuss power more in the next class.

Assumptions of the test It is often stated that a caveat of this test is that it assumes a *symmetric distribution of paired differences*. However, in practice this is not really a big limitation because under the null hypothesis that two groups are drawn from the same population, then the pairwise differences are guaranteed to be symmetrical, with both mean and median = 0.

Let's see if the distribution of paired differences for the triglyceride data looks approximately symmetrical:

```
tri_diff = post - pre
hist(tri_diff, breaks=10)
abline(v = median(tri_diff), lwd=2, col="blue")
abline(v = mean(tri_diff), lwd=2, col="red")
```



Well, these differences look sort of symmetrical, but not clearly so. However, what we are actually interested in here is symmetry under H_o, so we can use this test even when the actual paired differences are not symmetric about the mean/median.

The W statistic Instead of looking at the actual ***magnitudes*** of the paired differences between paired samples, here we will look at the ***ranks*** of the differences.

Our null hypothesis is that the ***sum of the ranks from both groups are the same***.

The steps are:

- Calculate the N differences between the pairs.
- Rank the *absolute* values of the n non-zero differences. Assign the average if there is a tie.
- Also record the sign of the differences, $sgn(x_{2i} - x_{1i})$ for $i \in \{1..N\}$.
- Compute the test statistic W, defined as:

$$W = \sum_{i=1}^{N} sgn(x_{2i} - x_{1i}) * R_i$$

```
# compute paired differences
tri_diff = post - pre
rank_diff = rank(abs(tri_diff))

# Calculate the Wilcoxon W statistic
w_stat = sum(sign(tri_diff) * rank_diff)
w_stat
```

```
## [1] -176
```

If there is no difference between the two samples, we would expect that $W = 0$, since there should be no bias toward lower or higher ranks in either sample.

The V statistic The R implementation of the Wilcoxon Rank Sum test uses something called a V-statistic, which is a little different than the W-statistic. The test will give the same p-value whether we use W or V, since the formulations are equivalent.

To compute V, instead of just adding up all the positive ranks and subtracting the negative ones to get our test statistic ($H_o : W = 0$), we just look at the positive and negative summed ranks separately. If both groups are the from the same distribution, then we'd expect these to be pretty much the same.

The test statistic, V, can range from 0 to $n(n+1)/2$. Depending on the question we want to ask, we choose V as follows:

- **2-tailed test**: $V = min(T_-, T_+)$, the minimum of the summed ranks with either sign
- **Upper-tailed test**: $V = T_-$
- **Lower-tailed test**: $V = T_+$

Let's go ahead and compute the V statistic and a corresponding p-value using the normal approximation for V:

```r
# negative rank sums
rank_diff[tri_diff < 0]
##  [1] 12 11 18 16 10 22 19 23  8  6  9  3  5 21 14 13  4 24
t_neg = sum(rank_diff[tri_diff < 0])
t_neg
## [1] 238

# positive rank sums
rank_diff[tri_diff > 0]
## [1]  1  7  2 17 15 20
t_pos = sum(rank_diff[tri_diff > 0])
t_pos
## [1] 62

# V statistic
v_stat = min(t_neg,t_pos)
v_stat
## [1] 62
```

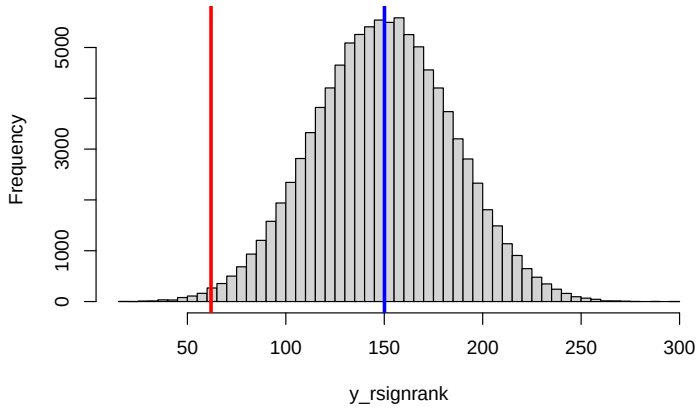
Sampling distribution of Wilcoxon statistics When the number of items sampled is more than ~20-25, the ***sampling distributions*** of the W and V statistics have a specific expected distribution under the null that is ***well approximated by a normal distribution***.

We can simulate the sampling distribution for V to illustrate this for ourselves. It turns out that R actually has a family of functions for the V distribution! To generate the samples, we will use the `rsignrank()` function, which takes random samples from a Wilcoxon Sign Rank distribution for a certain sample size n.

We will then plot a histogram of the sampling distribution with the mean overlaid in blue, and the value of our actual V-statistic overlaid in red.

```r
n=24  # size of triglyceride dataset

# sampling distribution of the test statistic W
set.seed(13000)                      # Set seed for reproducibility
N <- 100000                          # Specify sample size
y_rsignrank <- rsignrank(N, n = 24)  # Take N random samples from W(sample size=n)
head(y_rsignrank)                    # Print values to RStudio console
## [1] 167 109 199 145 205  95
hist(y_rsignrank,                    # Plot of randomly drawn density
     breaks = 50,
     main = "")
abline(v=mean(y_rsignrank), col="blue", lwd=3)
abline(v=mean(v_stat), col="red", lwd=3)
```



Normal approximations for the p-value In the absence of tied ranks (which we ignore for now to keep things simple), the population mean and variance of the W-statistic are:

$$\mu_W = 0;\ \sigma_W^2 = \frac{n(n+1)(2n+1)}{6}$$

And for the V-statistic, they are:

$$\mu_V = \frac{n(n+1)}{4}\ ;\ \ \sigma_V^2 = \frac{n(n+1)(2n+1)}{24}$$

For the triglyceride data, the sample size is $n = 24$. Let's use the normal approximations to compute a z-score and a p-value using both W and V:

```
# p-value (2-tailed) with normal approximation for W
n = length(tri_diff)
sigma_w = n*(n+1)*(2*n+1) / 6
z_w = w_stat / sqrt(sigma_w)
2*pnorm(z_w)
```

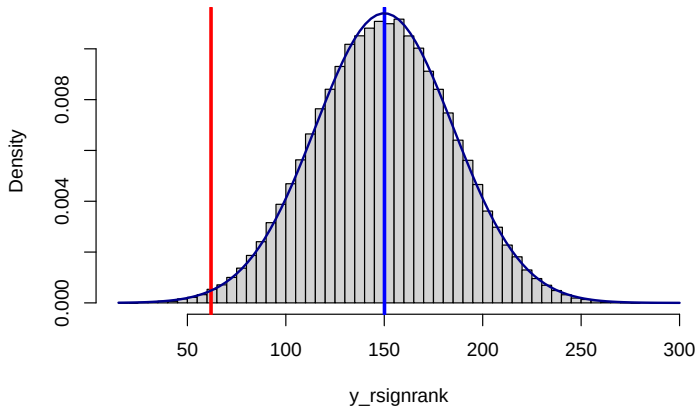
```
## [1] 0.01192738
```

```
# p-value (2-tailed) with normal approximation for V
mean_v = n*(n+1)/4              # mu = max_t / 2
sigma_v = n*(n+1)*(2*n+1) / 24  # without ties
z_v = (v_stat - mean_v) / sqrt(sigma_v)
2*pnorm(z_v)
```

```
## [1] 0.01192738
```

We can also remake our histogram above as a density plot, with a normal curve using the mean and SD for the V-distribution:

```
hist(y_rsignrank,
     breaks = 50, freq=FALSE,
     main = "")
abline(v=mean(y_rsignrank), col="blue", lwd=3)
abline(v=mean(v_stat), col="red", lwd=3)
curve(dnorm(x, mean_v, sqrt(sigma_v)),
      col="darkblue", lwd=2, add=TRUE)
```



This looks pretty good!

Wilcoxon Signed Rank Test in R To compute a p-value for our test statistic, we can use the `wilcox.test()` function with the `paired=TRUE` option to perform a paired test on the triglyceride data. The test gives the same result when samples are entered in either order.

```
wilcox.test(post, pre, paired=T)  # post vs. pre
```

```
##
##  Wilcoxon signed rank exact test
##
## data:  post and pre
## V = 62, p-value = 0.01051
## alternative hypothesis: true location shift is not equal to 0
```

```
wilcox.test(pre, post, paired=T)  # pre vs. post
```

```
##
##  Wilcoxon signed rank exact test
##
## data:  pre and post
## V = 238, p-value = 0.01051
## alternative hypothesis: true location shift is not equal to 0
```

Also, note that the paired two-sample test is equivalent to doing a one-sample test on the paired differences (thanks Omar for pointing this out!):

```r
# one-sample test using the paired differences
wilcox.test(tri_diff)
```

```
##
##  Wilcoxon signed rank exact test
##
## data:  tri_diff
## V = 62, p-value = 0.01051
## alternative hypothesis: true location is not equal to 0
```

Let's see what `psignrank()` gives us for a two-tailed probability:

```r
2*psignrank(v_stat, n)
```

```
## [1] 0.01050782
```

Notice that p-values returned by the paired test and the CDF function are the same, and both methods gave us an "exact" probability, as reported by the output of the `wilcox.test()`.

However, computing exact p-values for Wilcoxon statistics can become computationally expensive, especially for larger sample sizes, and particularly for the Rank Sum test for unpaired samples (see below).

By default, the `wilcoxtest()` will compute an exact p-value if the sample size is <50 and there are no ties. Otherwise, R will use a ***normal approximation*** with continuity correction to obtain a p-value.

Manual calculations vs. R functions How do the manual calculations for the p-value compare to those you got using the `wilcox.test()` and `psignrank()` functions in R? Why might these differ?

```r
# your answer here
The manual computation used the normal approximation whereas the R functions
compute exact probabilities, since n<50 and there are no ties.
```

Unpaired data: Mann-Whitney U / Wilcoxon Rank Sum test

What if our data are **not normal** and also **not paired**?

We can use a **Wilcoxon rank-sum** test or the equivalent **Mann-Whitney U-test**. Some people therefore like to refer to this test as a **Mann-Whitney-Wilcoxon** test.

This test is simlar to the paired signed-rank test, but instead of comparing ranks for the positive and negative differences between ***pairs*** of values, we will first combine the data, rank them together, and then compare how the summed ranks between the two ***independent groups***.

This makes intuitive sense because if the data are drawn from a homogeneous population, we would expect the ranks of the measurements from two random samples to be relatively well interleaved (as if we had shuffled a deck of cards). So, we would expect that the overall sum of ranks should be about the same for both groups.

The hypotheses to be tested are:

H_o: The sample distributions are the same.

H_A: The sample distributions are NOT the same.

W-statistics

To perform the test, we first compute the W-statistic for both groups:

- Combine the data.
- Assign ranks from smallest (top-ranked) to largest (lowest-ranked).
- Assign ties the midrank (the average of the ranks).
- Compute the sum of ranks T_1 for Sample 1 and T_2 for Sample 2.
- Calculate W using the formulas below.

$$W_1 = T_1 - \frac{n_1(n_1+1)}{2}$$
$$W_2 = T_2 - \frac{n_2(n_2+1)}{2}$$

where n_1 and n_2 are the sizes of the two groups.

***Note**: The sum of ranks for a sequential set of numbers starting with 1 is $N(N+1)/2$. Check this out for yourself on some small sets of numbers (e.g. 1,2,3,4,5). Intuitively, then, if most the measurements from Sample 1 are smaller smaller than those from Sample 2, then the **minimum sum of ranks** for Sample 1 would be $R_1 = n_1(n_1+1)/2$.*

Computing the W-statistics Ranks and rank sums:

```
Data = c(pre, post)

n_pre = length(pre)
n_post = length(post)
n = n_pre  # here, n is the same for both sets

# rank the combined data
DataRank = rank(Data)
DataRank
##  [1] 28.0 40.0 24.0 43.0 38.0 29.0 44.0 37.0 48.0 31.0 36.0  2.0 21.0 19.0  5.0
## [16] 32.5 30.0 45.0  7.0 39.0 35.0 27.0 47.0  8.0  9.0 42.0  6.0 22.5 12.0 10.0
## [31] 14.0  3.0 34.0 13.0 25.0 16.5 26.0  4.0  1.0 46.0 18.0 22.5 32.5 20.0 11.0
## [46] 16.5 15.0 41.0

DataRankSums = tapply(DataRank,rep(c("pre","post"),each=n),sum)
DataRankSums
##  post   pre
## 460.5 715.5
```

Test statistics:

```
t_pre  = DataRankSums["pre"]
t_post  = DataRankSums["post"]

t_min = n*(n + 1)/2      # here n and t_min are the same for both groups

w_pre = t_pre - t_min    # or t_pre_min = n_pre*(n_pre + 1)/2
w_pre
```

```
##   pre
## 415.5
w_post = t_post - t_min  # or t_post_min = n_post*(n_post + 1)/2
w_post
##  post
## 160.5
```

Normal approximation Again, the sampling distribution of W for samples $>$ ~20-25 each is approximately normal, so we can compute a p-value for our test statistic using a normal approximation.

The population mean and variance (again, ignoring the term for ties) are:

$$\mu_W = \frac{n_1 n_2}{2} \; ; \quad \sigma_W^2 = \frac{n_1 n_2}{12}(n_1 + n_2 + 1)$$

Now we can calculate a z-score and use it to get a p-value. By convention, we use the group with the lower W (here that is `w_post`):

```
# mean and SD for W sampling distribution
mu_w = n_pre * n_post / 2
mu_w
## [1] 288

sd_w_squared = (n_pre*n_post/12) * (n_pre + n_post + 1)
sd_w = sqrt(sd_w_squared)
sd_w
## [1] 48.49742

# z-score
z_w = (w_post-mu_w)/sd_w
z_w
##      post
## -2.629006

# p-value
2*pnorm(z_w, lower.tail = T)
##        post
## 0.008563493
2*pnorm(w_post, mean=mu_w, sd=sd_w, lower.tail = T)
##        post
## 0.008563493
```

Note that using the upper-tail probability for the higher W gives the same p-value:

```
# z-score
z_w = (w_pre - mu_w)/sd_w
z_w
##      pre
## 2.629006

# p-value (since the sign of the z-score is reversed, we use the opposite tail)
2*pnorm(z_w, lower.tail = F)
##         pre
```

```
## 0.008563493
2*pnorm(w_pre, mean=mu_w, sd=sd_w, lower.tail = F)
##         pre
## 0.008563493
```

U-statistics

The Mann-Whitney U-test uses a U-statistic instead, which measures the difference between the observed and minimum ranks. Mathematically, it is the same as the W-statistic, except the formulation is slightly different. The test statistic is the smaller of the rank sums.

$$U_1 = n_1 n_2 + \frac{n_1(n_1+1)}{2} - T_1$$
$$U_2 = n_1 n_2 - U_1$$

where n_1 and n_2 are the size of the two groups.

The population mean and variance (again, ignoring the term for ties for now) are:

$$\mu_U = \frac{n_1 n_2}{2} \; ; \; \sigma_U^2 = \frac{n_1 n_2}{12}(n_1 + n_2 + 1)$$

First, let's compute the U statistics by hand:

```
# see above for calculation of t_pre and t_post

t_min = n*(n+1)/2 # minimum rank

u_pre = n^2 + t_min - t_pre
u_pre
##   pre
## 160.5

u_post = n^2 - u_pre
u_post
##   pre
## 415.5
```

You can see that the values for the U-statistics are the same as the W-statistics, so using the normal approximation to compute the z-score and p-value will give exactly the same results as above using W-statistics!

Wilcoxon Rank Sum test in R

The test is implemented in R as `wilcox.test()`, with the (default) unpaired option. The p-value is the same when the groups are supplied in either order.

```
wilcox.test(pre, post)  # paired=FALSE is the default
```

```
## Warning in wilcox.test.default(pre, post): cannot compute exact p-value with
## ties
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  pre and post
## W = 415.5, p-value = 0.008821
## alternative hypothesis: true location shift is not equal to 0
```

```
wilcox.test(post, pre)  # the p-value is the same in either orientation
```

```
## Warning in wilcox.test.default(post, pre): cannot compute exact p-value with
## ties
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  post and pre
## W = 160.5, p-value = 0.008821
## alternative hypothesis: true location shift is not equal to 0
```

R also has the `wilcox` family of functions for the **distribution of the Wilcoxon Rank Sum statistic**. Since this is a discrete distribution, the presence of ties in the data makes the p-values inexact, as non-integer values are simply truncated.

```
w_pre
##   pre
## 415.5
w_post
##  post
## 160.5
```

```
## unadjusted values
2*pwilcox(w_pre, n_pre, n_post, lower.tail=F)       # truncates to 415
##         pre
## 0.007712559
2*pwilcox(w_post, n_pre, n_post)                    # truncates to 160
##        post
## 0.007712559
```

```
## adjusted values
# p(X => x): need to subtract 1 or else get p(X > x)
2*pwilcox(w_pre - 1.5, n_pre, n_post, lower.tail=F) # decreases to 414
##         pre
## 0.008230024

# p(X <= x)
2*pwilcox(w_post + 0.5, n_pre, n_post)              # increases to 161
##        post
## 0.008230024
```

The R documentation warns that the `wilcox` distribution functions are computationally inefficient and can sometimes crash R (!), so it is probably not a good idea to use them in general.

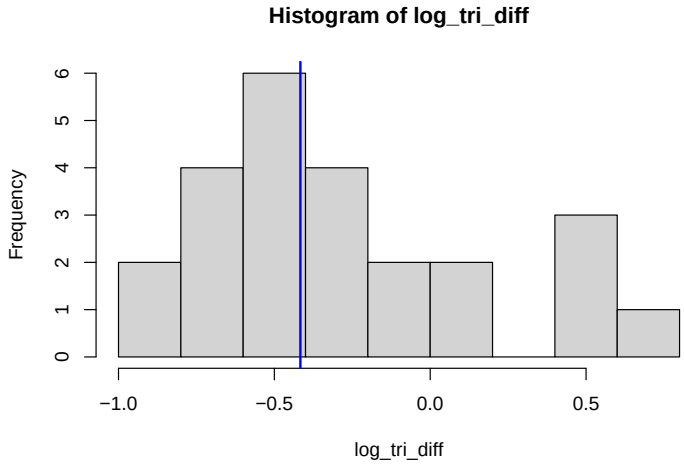
Manual calculations vs. R functions How do the manual results compare with the results from the R functions? Why are they not identical?

```
The manual calculations use the normal approximation; the test wants to perform
an exact test, but because of ties it has to use a normal approximation with
continuity correction.
```

What if we were to use the log-transformed data?

Try this out and see what you find. First, draw a histogram of the transformed data.

```
log_tri_diff = log(post) - log(pre)
hist(log_tri_diff, breaks=10)
abline(v = median(log_tri_diff), lwd=2, col="blue")
```



Now perform the test in R.

```
wilcox.test(pre,post)
```

```
## Warning in wilcox.test.default(pre, post): cannot compute exact p-value with
## ties
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  pre and post
## W = 415.5, p-value = 0.008821
## alternative hypothesis: true location shift is not equal to 0
```

```
wilcox.test(log(pre),log(post))
```

```
## Warning in wilcox.test.default(log(pre), log(post)): cannot compute exact p-
## value with ties
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  log(pre) and log(post)
## W = 415.5, p-value = 0.008821
## alternative hypothesis: true location shift is not equal to 0
```

How do the results compare with those for the untransformed data?

```
# your answer here
The differences in the log-transformed data actually do not look symmetrical
about the mean difference. The p-value nevertheless comes out the same because
the ranks do not change!

So the test still works pretty well even when the assumption of a symmetric
distribution of mean differences does not hold that well.
```

Permutation (shuffle) test [1]

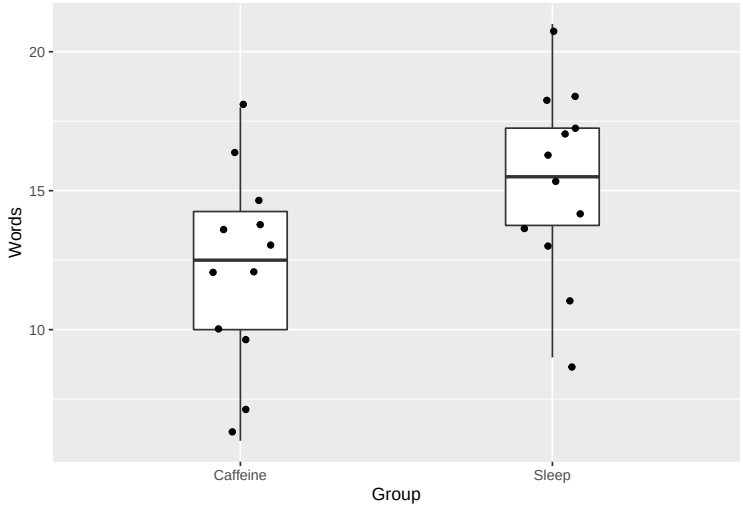
The `mosaic` package contains some convenient functions for sampling. The `Sleep` (capital S) dataset is included in this package. Here is the description of the study from the R documentation for this dataset:

In an experiment on memory (Mednicj et al, 2008), students were given lists of 24 words to memorize. After hearing the words they were assigned at random to different groups. One group of 12 students took a nap for 1.5 hours while a second group of 12 students stayed awake and was given a caffeine pill. The data set records the number of words each participant was able to recall after the break.

Let's make a boxplot of these data and take a look at them.

```
data(Sleep)

ggplot(Sleep, aes(x=Group, y=Words)) +
   geom_boxplot(width=0.3) +
   geom_jitter(position=position_jitter(0.1))
```



[1] This example is from: https://cran.r-project.org/web/packages/mosaic/vignettes/Resampling.html

If the two samples came from the same population, we would expect that if we jumble up the labels on the data points, then the mean difference between them would be zero.

We can do this many times and then find a p-value for the observed difference between the two groups.

This method uses **permutation** to randomly reassign the labels. Because this essentially has the effect of shuffling a deck of cards, the test is often called a **shuffle** test.

Let's try this out below. First, let's look at the difference in the means and then perform a regular t-test on them.

Note that here we use the **formula** syntax for the `mean()` and `t.test()` functions, which uses a tilde (`~`) symbol. This is a convenient notation to specify a relationship between two variables. The "dependent" variable (the outcome) goes on the left, and the independent variable goes on the right.

```
# scramble Group with respect to outcome, Words
mean(Words ~ Group, data = Sleep)              # means of the two samples
## Caffeine    Sleep
##    12.25    15.25
obs = diff(mean(Words ~ Group, data = Sleep))  # observed difference
obs
## Sleep
##     3
```

```
# t-test
t.test(Words ~ Group, Sleep, alternative="less") # test caffeine < sleep
```

```
##
##  Welch Two Sample t-test
##
## data:  Words by Group
## t = -2.1438, df = 21.894, p-value = 0.02171
## alternative hypothesis: true difference in means is less than 0
## 95 percent confidence interval:
##        -Inf -0.5965084
## sample estimates:
## mean in group Caffeine    mean in group Sleep
##                  12.25                  15.25
```

Looks like getting some sleep is better for your memory than taking caffeine!

If we randomly shuffle the group labels, we will get a different mean between the sample groups every time. We use the `shuffle()` function to do the permutation:

```
# one shuffle
diff(mean(Words ~ shuffle(Group), data = Sleep))
```

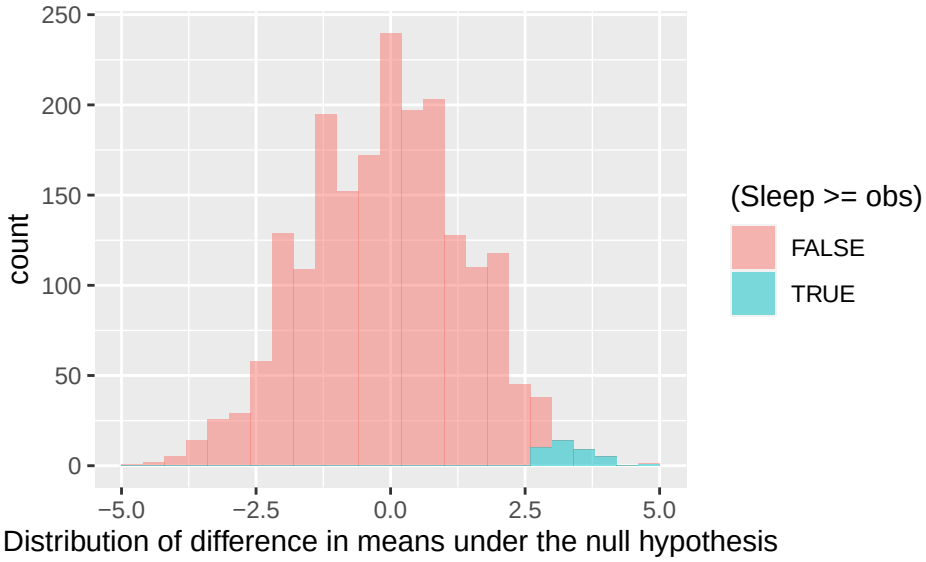
```
##     Sleep
## 0.6666667
```

Now let's try the permutation test and look at the results.

```r
# permutation test: scramble Group with respect to outcome, Words

# 2,000 shuffles
sleep_null <- do(2000) * diff(mean(Words ~ shuffle(Group), data = Sleep))

# from ggformula package, included with mosaic, enables formula-style expressions in ggplot
# here we just have one variable, but we can also use this to look at two variables
gf_histogram(gformula = ~ Sleep, fill = ~ (Sleep >= obs), data = sleep_null,
  binwidth = 0.4,
  xlab = "Distribution of difference in means under the null hypothesis")
```




```r
# empirical p-value
sum(sleep_null >= obs) / 2000
```

```
## [1] 0.0195
```

This is our empirical p-value for the difference in the sample means! How does it compare to the results of the t-test above?