ALTERNATIVE TESTS

1. **Wilcoxon rank-sum test:**

To do Wilcoxon rank-sum test in R, use:

* *wilcox.test(data~group, conf.int=T)* if your dataset has data 1 in one column and data 2 in another column
* *wilcox.test(data1, data2 , conf.int=T)* if your dataset has data 1 in one column and data 2 in another column

This test will result a two-tailed test result by default. The option “*conf.int*” will return the confidence interval as well. Also, the default is always 0.05 significant level (or 95% confidence interval). To change the significant level (or the confidence level), use:

*wilcox.test(data~group, conf.int=T, conf.level = your choice)*

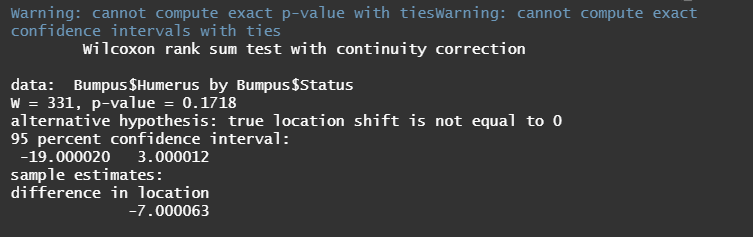


Figure 1: Output of a Wilcoxon rank sum test. Note that when there are ties in the data, the exact p value cannot be computed. In this case, set “*exact=F*” so that R uses normal approximation.

*wilcox.test(data~group, conf.int=T, exact=F)*

Example 1: As evidence in support of natural selection, the biologist Hermon Bumpus (google his name, if you like, for more background info) obtained measurements on adult male house sparrows after an uncommonly severe winter storm. Some of these birds had survived and some had perished. The humerus length of each bird was measured. The data can be accessed from the package Sleuth2 (“case0201”). As you already plotted the histograms of this data set in Assignment 1, the histograms are skewed. Use a rank-sum test to compare the two groups with .

1. **Wilcoxon signed-rank test:**

To do match paired Wilcoxon signed-rank test in R, use the command:

*wilcox.test(data\_1, data\_2, paired= T, conf.int=T)*

This test will result a two-tailed test result by default. Also, the default is always 0.05 significant level (or 95% confidence interval). To change the significant level (or the confidence level), use “*conf.level*”

Example 2: Darwin carried out an experiment to study whether seedlings from cross- fertilized plants tend to be superior to those from self-fertilized plants. He covered a number of plants with fine netting so that insects would be unable to fertilize them. He fertilized a number of flowers on each plant with their own pollen and he fertilized an equal number of flowers on the same plant with pollen from a distant plant. The seeds from the flowers were allowed to ripen and were set in wet sand to germinate. He placed two seedlings of the same age in a pot, one from a seed from a self-fertilized plant and one from a seed from a cross-fertilized flower. The heights of the plants were measured at a certain point in time (the data file is provided separately). Test the no treatment effect hypothesis using the Wilcoxon signed-rank test with . The data is provided in the file “DarwinData.csv.”

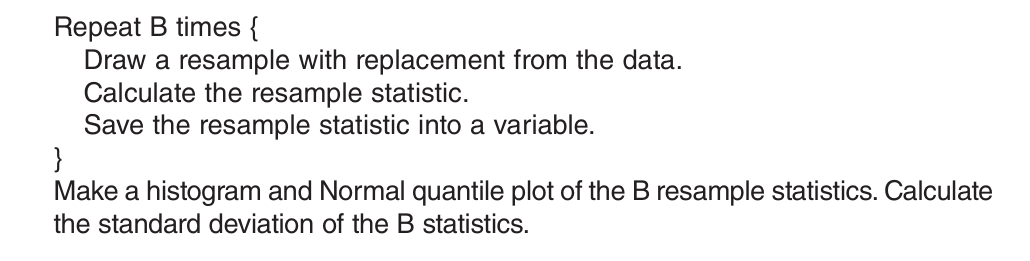
1. **Bootstrapping (The long way)**

As you generate data in R, you will get different results every time because it is random. If you want to get the same result every time, you will need to set the seed in R. And you should do that in every code chunk used for data generation:

*set.seed(seed)*

You can technically put in any number for seed.

The algorithm for bootstrapping is given below



Example 3: The original data is the time looking at a Facebook profile (provided in “FacebookData.csv”). Write an R code to create the bootstrap distribution by finding the mean for the 3,000 resamples. The draw the histogram and the QQ plot of this bootstrap distribution.

Note you will need to write a “for” loop for this. To write “for” loop in R, use:

*for (i in 1:B) {*

*……….. Your code here*

*}*

To do resample in R, use:

*sample (Name of the original data, size= n, replace = T or F)*

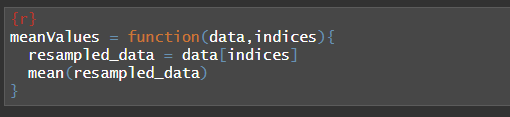
Here, size is the size of the data, replace =T if you want with replacement and F if you want without replacement.

1. **Bootstrapping (The short way)**

You can use *library(boot)* to automate bootstrapping in R.

*boot(Name of the original data, statistic, R= numbers of bootstrap samples)*

For the statistics argument here, you will need to write a function for this argument. In this case, we want to bootstrap and find the mean values. Thus, you can use the following function.



Then, you can call boot function as

*boot(Name of the original data, statistic=meanValues, R= numbers of bootstrap samples)*

Example 3 (Cont’d): Redo the bootstrapping above using the boot function. Look at your output in R and try to understand the lists within the output. Then, draw the histogram and the QQ plot of the bootstrapping distribution.

Example 4: Compute the 99% confidence interval of bootstrapping above using percentile method and BCa method.

To compute the confidence interval, use:

*boot.ci(Name of your boot output above, conf = …., type = “…”)*

Here, percentile is *type= “perc”* and BCa is *type= “bca”*. You can also do *type = “all”* to get other CI’s as well.

1. **Permutation test**

To do permutation test in R, you can either write your own code or use the built-in R functions. To use the built-in R functions, you need *library(perm)*.

To do permutation test, use:

*permTS(data1, data2, alternative=c("greater", “less”, “two.sided” ,method="exact.mc", control=permControl(nmc=N))*

Here *nmc=N* tells R how many permutations you want to do. We want to do at least 1000.

There are other methods to do permutation tests in R. For more information, check this out: <https://www.cfholbert.com/blog/two_sample_permutation_test/>.

Example 5: Do Example 1 in the lecture note (04\_PermutationTests) using R. The data is provided in “DPRData.csv” file. Choose N=5000.

Example 5 (cont’d): Do Example 1 the lecture note (04\_PermutationTests) but you will need to code it from scratch. Then, draw the histogram of the permutation distribution and compute the p value. Go to slides 4 and 5 of the lecture note (04\_PermutationTests) to review the steps (algorithm) to help with your coding. Note, you will need to use for loop and the sample command in R. Choose N=5000.