# IPS9 in R: Nonparametric tests (Chapter 15)

*Shukry Zablah (szablah20@amherst.edu) and Nicholas Horton (nhorton@amherst.edu)*

*July 22, 2018*

## Introduction and background

These documents are intended to help describe how to undertake analyses introduced as examples in the Ninth Edition of *Introduction to the Practice of Statistics* (2017) by Moore, McCabe, and Craig.

More information about the book can be found here. The data used in these documents can be found under Data Sets in the Student Site. This file as well as the associated R Markdown reproducible analysis source file used to create it can be found at https://nhorton.people.amherst.edu/ips9/.

This work leverages initiatives undertaken by Project MOSAIC (http://www.mosaic-web.org), an NSF-funded effort to improve the teaching of statistics, calculus, science and computing in the undergraduate curriculum. In particular, we utilize the `mosaic` package, which was written to simplify the use of R for introductory statistics courses. A short summary of the R needed to teach introductory statistics can be found in the mosaic package vignettes (http://cran.r-project.org/web/packages/mosaic). A paper describing the mosaic approach was published in the *R Journal*: https://journal.r-project.org/archive/2017/RJ-2017-024.

## Chapter 15: Nonparametric tests

This file replicates the analyses from Chapter 15: Nonparametric tests.

First, load the packages that will be needed for this document:

```
library(mosaic)
library(readr)
library(tidyr)
```

The skills in the this chapter will help analyze data that doesn't follow the Normal distribution and is not fit for the tests we previously looked at.

### Section 15.1: The Wilcoxon rank sum test

Let's read in the csv file for the Hits data in Example 15.1.

```
Hits <- read_csv("https://nhorton.people.amherst.edu/ips9/data/chapter15/EG15-001HITS.csv")
Hits
```

```
## # A tibble: 8 x 5
##   League   LeagueN  Hits HitsAmer HitsNat
##   <chr>      <int> <int>    <int>   <int>
## 1 American       1    21       21      19
## 2 American       1    18       18       7
## 3 American       1    24       24      11
## 4 American       1    20       20      13
## 5 National       2    19       NA      NA
## 6 National       2     7       NA      NA
## 7 National       2    11       NA      NA
## 8 National       2    13       NA      NA
```

To recreate the table shown, just select the two relevant columns for the hits of each group and drop the `NA` values.

```
Hits %>%
  select(HitsAmer, HitsNat) %>%
  tidyr::drop_na()
```

```
## # A tibble: 4 x 2
##   HitsAmer HitsNat
##      <int>   <int>
## 1       21      19
## 2       18       7
## 3       24      11
## 4       20      13
```

Dealing with `NA` values inside our observations can take up uneccesary space. Use the `tidyr::drop_na()` function to remove observations containing any `NA` values from your dataset.

Now let's set up the rank transformation.

```
nrows <- nrow(Hits)
RanksHits <- Hits %>%
  select(League, Hits) %>%
  arrange(Hits) %>%
  mutate(Rank = 1:nrows)
RanksHits
```

```
## # A tibble: 8 x 3
##   League    Hits  Rank
##   <chr>    <int> <int>
## 1 National     7     1
## 2 National    11     2
## 3 National    13     3
## 4 American    18     4
## 5 National    19     5
## 6 American    20     6
## 7 American    21     7
## 8 American    24     8
```

Next we sum the ranks for each group. We will take advantage of the `group_by()` and `summary()` idiom.

```
RanksHits %>%
  group_by(League) %>%
  summarize(`Sum of ranks` = sum(Rank))
```

```
## # A tibble: 2 x 2
##   League   `Sum of ranks`
##   <chr>             <int>
## 1 American             25
## 2 National             11
```

The intuition behind the test is that if there was no difference between the leagues both sums would be the same.

Finally, to perform the Wilcoxon Rank Sums test we use `wilcox.test()`.

```
wilcox.test(Hits ~ League, data = Hits)
```

```
##
##  Wilcoxon rank sum test
```

```
##
## data:  Hits by League
## W = 15, p-value = 0.05714
## alternative hypothesis: true location shift is not equal to 0
```

In Example 15.6 the book discusses how to deal with ties in the dataset (remember we rank the observations).

We have to recreate the dataset and confirm we can get the table of counts.

```
Exerg <- rbind(
  do(6) * data.frame(Exergamer = "Yes", TV_time = "None"),
  do(160) * data.frame(Exergamer= "Yes", TV_time = "<2 hours"),
  do(115) * data.frame(Exergamer = "Yes", TV_time = ">=2 hours"),
  do(48) * data.frame(Exergamer = "No", TV_time = "None"),
  do(616) * data.frame(Exergamer = "No", TV_time = "<2 hours"),
  do(255) * data.frame(Exergamer = "No", TV_time = ">=2 hours")
)
```

```
tally(~ Exergamer + TV_time, data = Exerg)
```

```
##          TV_time
## Exergamer None <2 hours >=2 hours
##       Yes    6      160       115
##       No    48      616       255
```

```
tally(~ Exergamer + TV_time, data = Exerg, format = "percent", margins = TRUE)
```

```
##          TV_time
## Exergamer       None   <2 hours  >=2 hours      Total
##     Yes     0.500000  13.333333   9.583333  23.416667
##     No      4.000000  51.333333  21.250000  76.583333
##     Total   4.500000  64.666667  30.833333 100.000000
```

Sucess! Now let's perform the Wilcox Rank Sum Test. R will take care of the fact that there are "ties" in the dataset. However, we do have to code the variables into numerics in order for the test to run.

```
Exerg %>%
  mutate(Val = recode(TV_time,
                      `None` = 0,
                      `<2 hours` = 1,
                      `>=2 hours` = 2)
         ) %>%
  wilcox.test(Val ~ Exergamer, data = .)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  Val by Exergamer
## W = 148130, p-value = 7.898e-06
## alternative hypothesis: true location shift is not equal to 0
```

### Section 15.2: The Wilcoxon signed rank test

Read in the csv file.

```
Story <- read_csv("https://nhorton.people.amherst.edu/ips9/data/chapter15/EG15-009STORY.csv")
Story
```

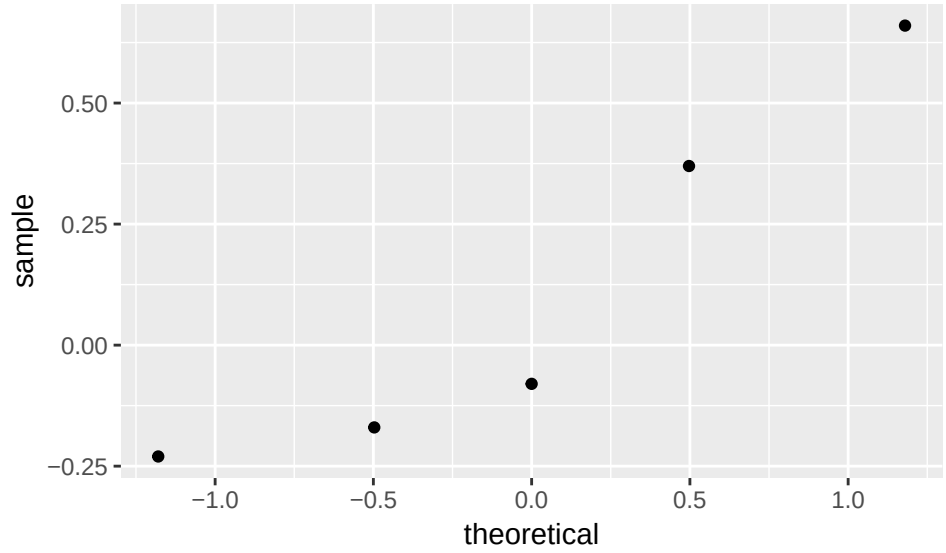
```
## # A tibble: 10 x 6
##    Child Progress Story1 Story2 DiffLow DiffHi
##    <int> <chr>     <dbl>  <dbl>   <dbl>  <dbl>
##  1     1 High      0.55    0.8    NA      0.25
##  2     2 High      0.570   0.82   NA      0.25
##  3     3 High      0.72    0.54   NA     -0.18
##  4     4 High      0.7     0.79   NA      0.09
##  5     5 High      0.84    0.89   NA      0.05
##  6     6 Low       0.4     0.77    0.37  NA
##  7     7 Low       0.72    0.49   -0.23  NA
##  8     8 Low       0       0.66    0.66  NA
##  9     9 Low       0.36    0.28   -0.08  NA
## 10    10 Low       0.55    0.38   -0.17  NA
```

Filter the relevant observations by using `filter()`.

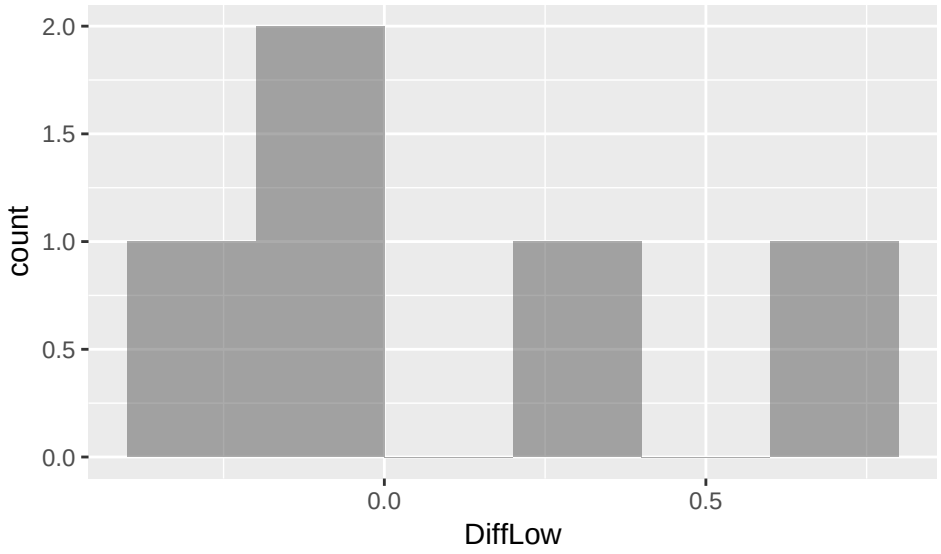
```
StoryTable <- Story %>%
  filter(Progress == "Low")
```

We will prove why our previous tests are not suitable for the scenario.

```
gf_qq(~ DiffLow, data = StoryTable)
```




```
gf_histogram(~ DiffLow, data = StoryTable, binwidth = 0.2, center = 0.5)
```




The plots suggest lack of Normality. This is why we use a rank test.

Before we do the test we have to format our dataset differently. Specifically, we need a column with the story type that has two levels. To do this we will use the `gather()` function that takes the names for the the two new columns that will be created and then a list of columns in the dataset that will make up the new ones. In this case we combine Story1 and Story2 into one column. Each observation in this column has the type of story attached to it.

The same `wilcox.test()` function will be used to perform the test. However, we specify `paired = TRUE` in order to perform the signed rank test from the book. The option `alternative = "less"` gets the one sided p-value that we are testing for.

```
StoryTableNarrower <- StoryTable %>%
  gather(key = "StoryType", value = "Values", Story1, Story2)
StoryTableNarrower
```

```
## # A tibble: 10 x 6
##    Child Progress DiffLow DiffHi StoryType Values
##    <int> <chr>      <dbl>  <dbl> <chr>      <dbl>
##  1     6 Low         0.37     NA Story1      0.4
##  2     7 Low        -0.23     NA Story1      0.72
##  3     8 Low         0.66     NA Story1      0
##  4     9 Low        -0.08     NA Story1      0.36
##  5    10 Low        -0.17     NA Story1      0.55
##  6     6 Low         0.37     NA Story2      0.77
##  7     7 Low        -0.23     NA Story2      0.49
##  8     8 Low         0.66     NA Story2      0.66
##  9     9 Low        -0.08     NA Story2      0.28
## 10    10 Low        -0.17     NA Story2      0.38
```

```
wilcox.test(Values ~ StoryType, data = StoryTableNarrower, paired = TRUE, alternative = "less")
```

```
##
##  Wilcoxon signed rank test
##
## data:  Values by StoryType
## V = 6, p-value = 0.4062
## alternative hypothesis: true location shift is less than 0
```

Example 15.12 uses the same function as before.

### Section 15.3: The Kruskal-Wallis test

The ANOVA test needs strict conditions to hold. The Kruskal-Wallis test provides an alternative to the one-way ANOVA test.

Let's read in the dataset.

```
Weeds <- read_csv("https://nhorton.people.amherst.edu/ips9/data/chapter15/EG15-014WEEDS.csv")
Weeds
```

```
## # A tibble: 16 x 2
##    weeds yield
##    <int> <dbl>
##  1     0  167.
##  2     0  172.
##  3     0  165
##  4     0  177.
##  5     1  166.
##  6     1  157.
##  7     1  167.
##  8     1  161.
##  9     3  159.
## 10     3  176.
## 11     3  153.
## 12     3  156
## 13     9  163.
## 14     9  142.
## 15     9  163.
## 16     9  162.
```

Recreate the summary statistics.

```
favstats(yield ~ weeds, data = Weeds)
```

```
##   weeds   min      Q1 median      Q3   max    mean        sd n missing
## 1     0 165.0 166.275 169.45 173.375 176.9 170.200  5.421562 4       0
## 2     1 157.3 160.150 163.65 166.325 166.7 162.825  4.468687 4       0
## 3     3 153.1 155.275 157.30 163.050 176.4 161.025 10.493292 4       0
## 4     9 142.4 157.400 162.55 162.725 162.8 157.575 10.118094 4       0
```

And get the test results! You can use `kruskal.test()` to perform the Kruskal-Wallis Rank Sum test.

```
kruskal.test(yield ~ weeds, data = Weeds)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  yield by weeds
## Kruskal-Wallis chi-squared = 5.5725, df = 3, p-value = 0.1344
```

These functions can be black boxes unless you understand the underlying process. Be sure to understand both the test procedure and know about the function in R. Type `?kruskal.test()` to find out more about the function. (You can do this for every function).

Let's look at another dataset.

```
Organic <- read_csv("https://nhorton.people.amherst.edu/ips9/data/chapter15/EG15-016ORGANIC.csv")
```

```
favstats(Score ~ Food, data = Organic)
```

```
##      Food  min     Q1 median     Q3  max     mean        sd  n missing
## 1 Comfort 3.67 4.5425  4.750 5.1700 6.00 4.887273 0.5729139 22       0
## 2 Control 3.83 4.7900  5.330 5.3725 6.17 5.082500 0.6216690 20       0
## 3 Organic 4.67 5.0000  5.585 6.1700 6.33 5.583500 0.5935644 20       0
```

And perform the same test.

```
kruskal.test(Score ~ as.factor(Food), data = Organic)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  Score by as.factor(Food)
## Kruskal-Wallis chi-squared = 12.409, df = 2, p-value = 0.00202
```

Note that the function won't work if the variable that has your groups is of type `char`. To fix this wrap the `as.factor()` function around it.