Quick Start when calculating MF from Hierarchical Data

Center for Veterinary Biologics - Statistics Section

December 21, 2018

Contents

INTRODUCTION

This document is intended to supplement function help pages and guide users through the steps of applying functions of the `MF` package to the evaluation of mitigated fraction when data is arranged in a nested hierarchical structure, first implemented in version 4.3.5. Details of the algorithm are covered in a separate vignette titled Algorithms for calculating MF from hierarchical data. When calculating MF for clustered but not hierarchical data, refer to help pages for functions `MFClus` and `MFClusBoot`. For convenience, all help pages can also be accessed via the MF Manual.

Data assumptions. Nested Hierarchical structure.

Examples in this document expect input data to be structured in a *nested hierarchical tree*, as shown in Table 1. If there is only one grouping variable or the experimental design is crossed, it is not appropriate to use this guide as written.

A nested design assumes that the factor level of one variable co-occurs with that of another variable. For example, Pen D exists only within Room Z.

Mitigated Fraction. Bootstrapped and Calculated.

Users have the option of simply calculating the mitigated fraction or simultaneously bootstrapping the mitigated fraction (including confidence interval) *and* calculating the mitigated fraction. Examples of both options are shown in this document.

Technical requirements.

Examples in this manual were created using MF version 4.3.5, R version 3.4.4 (2018-03-15) on Windows. CVB has not tested code usage on other systems.

The package can be found online at: https://github.com/ABS-dev/MF/blob/master/README.md, including installation instructions. It is expected that users have passing familiarity with R code and usage, as CVB does not have resources to address training or IT issues that may occur at external organizations.

Bug reports can be submitted online at: https://github.com/ABS-dev/MF/issues. To expedite resolution, please include a minimal working example and refrain from using confidential data. Incomplete issues may be closed without further investigation. Do not include confidential data as issue tracking is visible to all.

Table 1: Nested hierarchical data structure.

room	pen	litter	tx	lung
Room W	Pen A	Litter 11	vac	5.63
				4.62
			con	9.20
				7.28
		Litter 12	vac	3.76
				3.86
			con	6.31
				3.52
	Pen B	Litter 13	vac	4.36
				6.10
			con	7.87
				8.34
		Litter 14	vac	5.39
				5.80
			con	8.28
				7.95
	Pen C	Litter 15	vac	4.86
				5.66
			con	7.66
				8.52
		Litter 16	vac	4.17
				4.84
			con	5.82
				7.72
Room Z	Pen D	Litter 17	vac	3.60
				4.92
			con	5.22
				5.93
		Litter 18	vac	7.62
				5.04
			con	8.62
				6.27
	Pen E	Litter 19	vac	3.79
				5.38
			con	9.45
				6.51
		Litter 20	vac	5.30
				4.42
			con	7.21
				6.35
	Pen F	Litter 21	vac	4.69
				6.04
			con	6.64
				7.00
		Litter 22	vac	4.90
				5.37
			con	6.77

CALCULATED MF

General use.

To calculate mitigated fraction directly from data without bootstrapping, use the function `MFClusHier`. This function requires four inputs:

- `formula`: The formula in form `y ~ x + a/b/c` where `y` is a continuous response, `x` is a factor with two levels of treatment, and `a/b/c` are grouping variables. Nesting is assumed to be in order, left to right, highest to lowest. So a single level of "a" will contain multiple levels of "b" and a single level of "b" will contain multiple levels of "c".
- `data`: The data table (`data.frame` or `tibble`) with variables as identified in formula. If there are extra variables, they will be ignored.
- `compare`: Treatment groups, a text vector of length two. The first position [1] is treated as the control or reference group to which members of the second position [2] are compared.
- `which.factor`: One or more variable(s) of interest. This can be any of the grouping variables from the data set. If "All" is specified, a summary MF will be calculated for the whole tree.

Refer to the `MFClusHier` help page (`?MFClusHier`) for extended usage details.

Looking at MF for levels of a variable.

The `which.factor` argument allows users the option of selecting if mitigated fraction should be calculated for levels of a particular variable. In the data from Table 1, it may be of interest to calculate the mitigated fraction for each level of variable "room". It is possible to consider levels from multiple variables simultaneously, and the designator `All` can be used to evaluate for the entire tree, without breaking out by any variable. Evaluating the value of mitigated fraction for the entire tree is the default behavior, should the user not specify anything to the `which.factor` argument.

Common coding examples.

Default case, looking only at whole tree:

```
MFClusHier(formula = lung ~ tx + room/pen/litter, data = a)
```

Selecting a single variable to evaluate for each factor:

```
MFClusHier(formula = lung ~ tx + room/pen/litter, data = a,
                      which.factor = 'room')
```

Selecting multiple variables, including the entire tree:

```
MFClusHier(formula = lung ~ tx + room/pen/litter, data = a,
                       which.factor = c('room', 'litter', 'All'))
```

Basic Output.

The default display of `MFClusHier` output is a table with one row for each unique mitigated fraction calculated. A unique mitigated fraction value will be calculated for each level of the variables identified by the user in the argument `which.factor` (see previous discussion). The total number of rows is the sum of the number of unique levels for each grouping variable the user specified. The value "All" will add one row to this table.

```
mf_multiple <- MFClusHier(formula = lung ~ tx + room/pen/litter, data = a,
                       which.factor = c('room', 'litter', 'All'))
mf_multiple
```

```
##    variable     level   MF N1N2  U con_N vac_N con_medResp vac_medResp
## 1      room    Room W 0.83   24 22    12    12       7.795       4.850
## 2      room    Room Z 0.91   22 21    11    12       6.640       4.980
## 3    litter Litter 11 1.00    4  4     2     2       8.240       5.125
## 4    litter Litter 12 0.00    4  2     2     2       4.915       3.810
## 5    litter Litter 13 1.00    4  4     2     2       8.105       5.230
## 6    litter Litter 14 1.00    4  4     2     2       8.115       5.595
## 7    litter Litter 15 1.00    4  4     2     2       8.090       5.260
## 8    litter Litter 16 1.00    4  4     2     2       6.770       4.505
## 9    litter Litter 17 1.00    4  4     2     2       5.575       4.260
## 10   litter Litter 18 0.50    4  3     2     2       7.445       6.330
## 11   litter Litter 19 1.00    4  4     2     2       7.980       4.585
## 12   litter Litter 20 1.00    4  4     2     2       6.780       4.860
## 13   litter Litter 21 1.00    4  4     2     2       6.820       5.365
## 14   litter Litter 22 1.00    2  2     1     2       6.770       5.135
## 15      All        All 0.87   46 43    23    24       7.210       4.910
```

The table columns are:

- `variable`: Which variable was considered when evaluating MF for the row.
- `level`: A unique factor level of the variable which was considered when evaluating MF for the row.
- `MF`: Mitigated fraction calculated value.
- `N1N2`: Sum of the `n1n2` values from the rank table for the factor level of that row.
- `U`: Sum of the `u` values from the rank table for the factor level of that row.
- `con_N` & `vac_N`: Sum of the counts from the rank table matching the particular factor level of the row. Note that the left hand side of the underscore will match values passed by user to `compare`.

- `con_medResp` & `vac_medResp`: Median of responses for each comparison group matching the particular factor level of the row. Note that the left hand side of the underscore will match values passed by user to argument `compare`.

Advanced usage.

This section covers the technical description of how to access the rank table from `MFh()` and summarize using `MFnest()`. For a full discussion of the algorithms used in these functions and how they are related, see Algorithms for calculating MF from hierarchical data.

Rank table.

To access the rank table, use the **MFh** field of the output object from `MFClusHier()`. For example:

```
thisMFh <- mf_multiple$MFh
thisMFh
```

```
##      room   pen    litter con_medResp con_n w vac_medResp vac_n n1n2 u
## 1  Room W Pen A Litter 11       8.240     2 7       5.125     2    4 4
## 2  Room W Pen A Litter 12       4.915     2 5       3.810     2    4 2
## 3  Room W Pen B Litter 13       8.105     2 7       5.230     2    4 4
## 4  Room W Pen B Litter 14       8.115     2 7       5.595     2    4 4
## 5  Room W Pen C Litter 15       8.090     2 7       5.260     2    4 4
## 6  Room W Pen C Litter 16       6.770     2 7       4.505     2    4 4
## 7  Room Z Pen D Litter 17       5.575     2 7       4.260     2    4 4
## 8  Room Z Pen D Litter 18       7.445     2 6       6.330     2    4 3
## 9  Room Z Pen E Litter 19       7.980     2 7       4.585     2    4 4
## 10 Room Z Pen E Litter 20       6.780     2 7       4.860     2    4 4
## 11 Room Z Pen F Litter 21       6.820     2 7       5.365     2    4 4
## 12 Room Z Pen F Litter 22       6.770     1 3       5.135     2    2 2
```

Alternatively, calculate rank table directly by using the `MFh()` function:

```
MFh(formula = lung ~ tx + room/pen/litter, data = a)
```

```
##      room   pen    litter con_medResp con_n w vac_medResp vac_n n1n2 u
## 1  Room W Pen A Litter 11       8.240     2 7       5.125     2    4 4
## 2  Room W Pen A Litter 12       4.915     2 5       3.810     2    4 2
## 3  Room W Pen B Litter 13       8.105     2 7       5.230     2    4 4
## 4  Room W Pen B Litter 14       8.115     2 7       5.595     2    4 4
## 5  Room W Pen C Litter 15       8.090     2 7       5.260     2    4 4
## 6  Room W Pen C Litter 16       6.770     2 7       4.505     2    4 4
## 7  Room Z Pen D Litter 17       5.575     2 7       4.260     2    4 4
## 8  Room Z Pen D Litter 18       7.445     2 6       6.330     2    4 3
```

```
## 9  Room Z Pen E Litter 19       7.980     2 7       4.585     2    4 4
## 10 Room Z Pen E Litter 20       6.780     2 7       4.860     2    4 4
## 11 Room Z Pen F Litter 21       6.820     2 7       5.365     2    4 4
## 12 Room Z Pen F Litter 22       6.770     1 3       5.135     2    2 2
```

The output table includes the following information, used to calculate the Basic Output table as described previously:

- `con_n` & `vac_n`: Counts of observations for each treatment for a particular instance of unique factor level of a variable. Note that the left hand side of the underscore will match values passed by user to `compare`.
- `n1n2`: Product of counts, `con_n` * `vac_n`.
- `w`: Wilcoxon statistic.
- `u`: Mann-Whitney statistic.
- `con_medResp` & `vac_medResp`: Median observed response for each treatment group in a particular instance of unique factor level. Note that the left hand side of the underscore will match values passed by user to `compare`.

There is one row for each unique combination of factor levels across all variables. This table shows the initial statistics as determined by the experimental design. Refer to the vignette for algorithm design Algorithms for calculating MF from hierarchical data for in-depth discussion of how the rank is used to evaluate mitigated fraction values.

The rank table is not affected by changes to the `which.factor` argument.

Reproducibility.

The rank table can be used to re-calculate the mitigated fraction values, for example if the user intends to explore a different selection to the `which.factor` argument. To do this, use the function `MFnest` which takes the following input arguments:

- `Y`: **MFh** field output as from `MFClusHier()`
- `which.factor`: As above.

For example:

```
MFnest(thisMFh, which.factor = "pen")
```

```
##   variable level   MF N1N2 U con_N vac_N con_medResp vac_medResp
## 1      pen Pen A 0.50    8 6     4     4       6.795       4.240
## 2      pen Pen B 1.00    8 8     4     4       8.115       5.595
## 3      pen Pen C 1.00    8 8     4     4       7.690       4.850
## 4      pen Pen D 0.75    8 7     4     4       6.100       4.980
## 5      pen Pen E 1.00    8 8     4     4       6.860       4.860
## 6      pen Pen F 1.00    6 6     3     4       6.770       5.135
```

This is the same output as if the user had initially selected for variable “pen”:

```
MFClusHier(formula = lung ~ tx + room/pen/litter, data = a,
                      which.factor = "pen")
```

```
##   variable level   MF N1N2 U con_N vac_N con_medResp vac_medResp
## 1      pen Pen A 0.50    8 6     4     4       6.795       4.240
## 2      pen Pen B 1.00    8 8     4     4       8.115       5.595
## 3      pen Pen C 1.00    8 8     4     4       7.690       4.850
## 4      pen Pen D 0.75    8 7     4     4       6.100       4.980
## 5      pen Pen E 1.00    8 8     4     4       6.860       4.860
## 6      pen Pen F 1.00    6 6     3     4       6.770       5.135
```

BOOTSTRAPPED MF

General use.

The function `MFClusBootHier` allows for a bootstrapping approach to calculating mitigated fraction values. Input arguments are:

Same as in non-bootstrapped usage (i.e. MFClusHier)

- `formula`: The formula in form `y ~ x + a/b/c` where `y` is a continuous response, `x` is a factor with two levels of treatment, and `a/b/c` are grouping variables. Nesting is assumed to be in order, left to right, highest to lowest. So a single level of "a" will contain multiple levels of "b" and a single level of "b" will contain multiple levels of "c".
- `data`: The data table (`data.frame` or `tibble`) with variables as identified in argument `formula`. If there are extra variables, they will be ignored.
- `compare`: Treatment groups, a text vector of length two. The first position [1] is treated as the control or reference group to which members of the second position [2] are compared.
- `which.factor`: Variable(s) of interest. This can be any of the grouping variables from the data set. If "All" is specified, a summary MF will be calculated for the whole tree.

Additional arguments

- `nboot`: Number of bootstrapping events.
- `boot.unit`: Boolean whether to sample observations from within those of the same core.
- `boot.cluster`: Boolean whether to sample which clusters are present. If TRUE, some trees have all the clusters represented in the original data while others only have a subset.
- `alpha`: Complement of the confidence level. As used in `MFClusBoot`.
- `seed`: Used to initialize random number generator for reproducibility.

A "core" is the unique combination of variable levels from the data, including the `compare` designation. In Table 1, one core would be Room W/Pen A/Litter 11/vac and another would

be Room Z/Pen F/Litter 22/con.

A "cluster" is the the unique combination of variable levels from the data, without the `compare` designation. In Table 1, the first four observations are from the same core, Room W/Pen A/Litter 11.

Refer to the `MFClusHier` help page (`?MFClusBootHier`) for extended usage details.

Further discussion regarding the bootstrapping algorithm can be found in Algorithms for calculating MF from hierarchical data.

Common coding examples.

Default case, looking only at whole tree and bootstrapping both at the cluster and unit levels:

```
MFClusBootHier(formula = lung ~ tx + room/pen/litter,
                               data = a)
```

Specifying what bootstrapping sampling to occur:

```
MFClusBootHier(formula = lung ~ tx + room/pen/litter,
                                data = a, boot.unit = TRUE,
                                boot.cluster = FALSE, which.factor = 'room')
```

Adjusting the number of bootstrapping events and alpha:

```
MFClusBootHier(formula = lung ~ tx + room/pen/litter,
                                    data = a, boot.unit = FALSE,
                                    boot.cluster = TRUE,  alpha = 0.1,
                                    which.factor = c('room', 'litter', 'All'))
```

Basic output.

The default display of `MFClusBootHier` is a table with one row for each unique mitigated fraction calculated, just like in the non-bootstrapped approach. However, instead of summary statistics about the calculated mitigated fraction, there are values summarizing the mitigated fraction from a bootstrapped population.

```
mfboot_multiple <- MFClusBootHier(formula = lung ~ tx + room/pen/litter,
                                  data = a, boot.unit = FALSE,
                                  boot.cluster = TRUE,  alpha = 0.1,
                                  which.factor = c('room', 'litter', 'All'),
                                  seed = 150)
mfboot_multiple
```

```
##    variable     level median etlower etupper hdlower hdupper mf.obs
## 1      room    Room W   0.83    0.50     1.0    0.60     1.0   0.83
## 2      room    Room Z   0.92    0.75     1.0    0.80     1.0   0.91
## 3    litter Litter 11   1.00    1.00     1.0    1.00     1.0   1.00
## 4    litter Litter 12   0.00    0.00     0.0    0.00     0.0   0.00
## 5    litter Litter 13   1.00    1.00     1.0    1.00     1.0   1.00
## 6    litter Litter 14   1.00    1.00     1.0    1.00     1.0   1.00
## 7    litter Litter 15   1.00    1.00     1.0    1.00     1.0   1.00
## 8    litter Litter 16   1.00    1.00     1.0    1.00     1.0   1.00
## 9    litter Litter 17   1.00    1.00     1.0    1.00     1.0   1.00
## 10   litter Litter 18   0.50    0.50     0.5    0.50     0.5   0.50
## 11   litter Litter 19   1.00    1.00     1.0    1.00     1.0   1.00
## 12   litter Litter 20   1.00    1.00     1.0    1.00     1.0   1.00
## 13   litter Litter 21   1.00    1.00     1.0    1.00     1.0   1.00
## 14   litter Litter 22   1.00    1.00     1.0    1.00     1.0   1.00
## 15      All        All   0.88    0.71     1.0    0.75     1.0   0.87
```

The variable columns are:

- `variable`: Which variable was considered when evaluating MF for the row.
- `level`: A unique factor level of the variable which was considered when evaluating MF for the row.
- `median`: Median of mitigated fractions calculated from the bootstrapped population.
- `etlower`: Lower value of equal tailed range of mitigated fractions calculated from the bootstrapped population.
- `etupper`: Upper value of equal tailed range of mitigated fractions calculated from the bootstrapped population.
- `hdlower`: Lower value of the highest posterior density range of mitigated fractions calculated from the bootstrapped population.
- `hdupper`: Upper value of the highest posterior density range of mitigated fractions calculated from the bootstrapped population.
- `mf.obs`: Mitigated fraction value calculated from data input, without bootstrapping.

Advanced usage.

This section covers the technical description of how to access the bootstrapped results and use the `MFnestBoot()` function. For a full discussion of the algorithms of how `MFhBoot()`, `MFnestBoot()` and `MFClusBootHier` are related, or details of the bootstrapping algorithm, see Algorithms for calculating MF from hierarchical data.

Bootstrapping step output.

The bootstrapping stage is the most computationally intensive, so a user may wish to bypass this step subsequently if the only change are values being passed to the `which.factor` or `alpha` arguments. Access bootstrapping step outputs using the **MFhBoot** field of the output object. For example:

```
thisBootMFh <- mfboot_multiple$MFhBoot
thisBootMFh
```

```
## $bootmfh
## # A tibble: 120,000 x 11
##    bootID con_medResp con_n     w vac_medResp vac_n     u  n1n2 room
##     <int>       <dbl> <dbl> <dbl>       <dbl> <dbl> <dbl> <dbl> <chr>
##  1      1        8.24     2     7        5.12     2     4     4 Room W
##  2      1        8.10     2     7        5.23     2     4     4 Room W
##  3      1        8.09     2     7        5.26     2     4     4 Room W
##  4      1        8.09     2     7        5.26     2     4     4 Room W
##  5      1        8.09     2     7        5.26     2     4     4 Room W
##  6      1        8.09     2     7        5.26     2     4     4 Room W
##  7      1        6.77     2     7        4.50     2     4     4 Room W
```

```
##  8      1        6.77     2     7        4.50     2     4     4 Room W
##  9      1        5.57     2     7        4.26     2     4     4 Room Z
## 10      1        6.82     2     7        5.36     2     4     4 Room Z
## # ... with 119,990 more rows, and 2 more variables: pen <chr>,
## #   litter <chr>
##
## $clusters
##      room   pen    litter clusterID
## 1  Room W Pen A Litter 11         1
## 2  Room W Pen A Litter 12         2
## 3  Room W Pen B Litter 13         3
## 4  Room W Pen B Litter 14         4
## 5  Room W Pen C Litter 15         5
## 6  Room W Pen C Litter 16         6
## 7  Room Z Pen D Litter 17         7
## 8  Room Z Pen D Litter 18         8
## 9  Room Z Pen E Litter 19         9
## 10 Room Z Pen E Litter 20        10
## 11 Room Z Pen F Litter 21        11
## 12 Room Z Pen F Litter 22        12
##
## $compare
## [1] "con" "vac"
##
## $mfh
## # A tibble: 12 x 10
##    room   pen   litter    con_medResp con_n     w vac_medResp vac_n  n1n2
##    <chr>  <chr> <chr>           <dbl> <dbl> <dbl>       <dbl> <dbl> <dbl>
##  1 Room W Pen A Litter 11        8.24     2     7        5.12     2     4
##  2 Room W Pen A Litter 12        4.92     2     5        3.81     2     4
##  3 Room W Pen B Litter 13        8.10     2     7        5.23     2     4
##  4 Room W Pen B Litter 14        8.12     2     7        5.60     2     4
##  5 Room W Pen C Litter 15        8.09     2     7        5.26     2     4
##  6 Room W Pen C Litter 16        6.77     2     7        4.50     2     4
##  7 Room Z Pen D Litter 17        5.57     2     7        4.26     2     4
##  8 Room Z Pen D Litter 18        7.44     2     6        6.33     2     4
##  9 Room Z Pen E Litter 19        7.98     2     7        4.58     2     4
## 10 Room Z Pen E Litter 20        6.78     2     7        4.86     2     4
## 11 Room Z Pen F Litter 21        6.82     2     7        5.36     2     4
## 12 Room Z Pen F Litter 22        6.77     1     3        5.14     2     2
## # ... with 1 more variable: u <dbl>
##
## $seed
## [1] 150
```

It is possible to calculate `mfboot_multiple$MFhBoot` directly by using the `MFhBoot()` function, however this executes the bootstrapping stage again. To get the same output with both approaches, the `seed` argument must be the same value.

```
MFhBoot(formula = lung ~ tx + room/pen/litter,
                                    data = a, boot.unit = FALSE,
                                    boot.cluster = TRUE,
                                    seed = 150)
```

This output is a list of four objects:

- `bootmfh`: The rank table of all the bootstrapped data sets. This is formatted as the rank table for non-bootstrapped data with designation of a single bootstrapping instance using the additional variable **bootID**. Values for **bootID** are simply `1:nboot`.
- `clusters`: Unique clusters in the data.
- `compare`: User-supplied value passed to compare argument.
- `mfh`: The rank table from user-supplied data; formatted as for non-bootstrapped data.

Note that in the particular case of `boot.cluster = FALSE`, although all `nboot` instances will be the same clusters as input data, observed responses will vary due to sampling of the cores. The case of both `boot.cluster` and `boot.unit` as FALSE is simply calculating MF from data without any bootstrapping.

Reproducibility.

To bootstrap from the same data in a reproducible manner, numerical value for the **seed** argument in `MFClusBootHier`. If no value for `seed` is specified, the function defaults to a value of `sample(1:1e+05, 1)` and subsequent iterations may yield different summaries.

```
MFClusBootHier(formula = lung ~ tx + room/pen/litter,
                                  data = a, seed = 150)
```

Since the bootstrapping step is the most computationally intensive, a user may wish to use the bootstrapped rank table to recalculate mitigated fraction for levels of different variables or a different `alpha` for calculating confidence intervals. This is possible using the `MFnestBoot` function, which takes the following input arguments:

- `x`: **MFhBoot** field output as from `MFClusBootHier`.
- `which.factor`: As above.
- `alpha`: As above.

For example:

```
MFnestBoot(thisBootMFh, which.factor = c('pen', 'All'), alpha = 0.1)
```

```
## Complete separation observed for variable(s): pen
```

```
## $mfnest_details
## # A tibble: 63,296 x 8
```

```
## # Groups:   variable, level [?]
##    variable level bootID     U  N1N2 con_N vac_N    MF
##    <chr>    <chr>  <int> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 All      All        1    46    46    23    24 1
##  2 All      All        2    44    46    23    24 0.913
##  3 All      All        3    44    46    23    24 0.913
##  4 All      All        4    46    48    24    24 0.917
##  5 All      All        5    42    44    22    24 0.909
##  6 All      All        6    43    44    22    24 0.955
##  7 All      All        7    44    46    23    24 0.913
##  8 All      All        8    42    46    23    24 0.826
##  9 All      All        9    44    48    24    24 0.833
## 10 All      All       10    42    46    23    24 0.826
## # ... with 63,286 more rows
##
## $mfnest_summary
## # A tibble: 7 x 8
##   variable level median etlower etupper hdlower hdupper mf.obs
##   <fct>    <chr>  <dbl>   <dbl>   <dbl>   <dbl>   <dbl>  <dbl>
## 1 pen      Pen A  0.5     0           1    0           1  0.5
## 2 pen      Pen B  1       1           1    1           1  1
## 3 pen      Pen C  1       1           1    1           1  1
## 4 pen      Pen D  0.75    0.5         1    0.5         1  0.75
## 5 pen      Pen E  1       1           1    1           1  1
## 6 pen      Pen F  1       1           1    1           1  1
## 7 All      All    0.875   0.708       1    0.75        1  0.870
##
## $seed
## [1] 150
```

APPENDIX

Code for example data.

```
a <- data.frame(
 room = paste('Room', rep(c('W', 'Z'), each = 24)),
 pen = paste('Pen', rep(LETTERS[1:6], each = 8)),
 litter = paste('Litter', rep(11:22, each = 4)),
 tx = rep(rep(c('vac', 'con'), each = 2), 12),
 stringsAsFactors = FALSE
 )
set.seed(76153)
```

```r
a$lung[a$tx == 'vac'] <- round(rnorm(24, 5, 1.3), 2)
a$lung[a$tx == 'con'] <- round(rnorm(24, 7, 1.3), 2)
a <- a[-48,]
```

Session details for this manual.

```r
sessionInfo()
```

```
## R version 3.4.4 (2018-03-15)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 14393)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] bindrcpp_0.2.2   MF_4.3.5         knitr_1.21       forcats_0.3.0
##  [5] stringr_1.3.1    dplyr_0.7.8      purrr_0.2.5      readr_1.1.1
##  [9] tidyr_0.8.2      tibble_1.4.2     ggplot2_3.1.0    tidyverse_1.2.1
## [13] kableExtra_0.9.0
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_0.2.5  xfun_0.3          haven_1.1.2
##  [4] lattice_0.20-35   colorspace_1.3-2  htmltools_0.3.6
##  [7] viridisLite_0.3.0 yaml_2.1.19       utf8_1.1.4
## [10] rlang_0.3.0.1     pillar_1.2.3      glue_1.3.0
## [13] withr_2.1.2       modelr_0.1.2      readxl_1.1.0
## [16] bindr_0.1.1       plyr_1.8.4        munsell_0.5.0
## [19] gtable_0.2.0      cellranger_1.1.0  rvest_0.3.2
## [22] evaluate_0.10.1   broom_0.5.0       Rcpp_1.0.0
## [25] scales_0.5.0      backports_1.1.2   jsonlite_1.5
## [28] hms_0.4.2         digest_0.6.15     stringi_1.1.7
## [31] grid_3.4.4        cli_1.0.0         tools_3.4.4
## [34] magrittr_1.5      lazyeval_0.2.1    crayon_1.3.4
```

```
## [37] pkgconfig_2.0.1   xml2_1.2.0         lubridate_1.7.4
## [40] assertthat_0.2.0  rmarkdown_1.11     httr_1.3.1
## [43] rstudioapi_0.7    R6_2.2.2           nlme_3.1-131.1
## [46] compiler_3.4.4
```