# Why the package mixed

## The problem

The R ecosystem already has some very good packages that deal with labelled objects. In particular, the inter-connected packages **haven** and **labelled** provide the functionality most users would ever need.

As nice and useful these packages are, it has become apparent they have some fundamental design features that run, in some situations, against users' expectations. This has a lot to do with the treatment of declared missing values, that are instrumental for the social sciences.

The following minimal example (adapted from the vignette in package **haven**) illustrates the situation:

```
> library(haven)
> x1 <- labelled_spss(c(1:5, 99), labels = c(Missing = 99), na_value = 99)
```

The printed objects from this package nicely display some properties:

```
> x1
<labelled_spss<double>[6]>
[1]  1  2  3  4  5 99
Missing values: 99

Labels:
 value   label
    99 Missing
```

There are 5 normal values (suppose they represent number of children), and one declared missing value coded 99. This value *acts* as a missing value, but it is different from a regular missing value in R, coded NA. The later stands for any missing information (something like an empty cell) regardless of the reason.

Here, on the other hand, the cell is *not* empty, but the value 99 is not a valid value either. It cannot possibly represent 99 children in the household, but for instance it could mean the respondent did not want to respond. It is properly identified as missing, with:

```
> is.na(x1)
[1] FALSE FALSE FALSE FALSE FALSE  TRUE
```

But when calculating a mean, for instance, the normal expectation is that value 99 would not play any role in the calculations (since it should be *missing*). However:

```
> mean(x1)
[1] 19
```

This means the value 99 did play an active role despite being identified as "missing". In an ideal world, the expected mean would be 3, or at best employ the argument `na.rm = TRUE` if the result is NA because of the declared missing value.

A solution to this problem is offered by package **labelled**, which has a function called `user_na_to_na()`:

```
> library(labelled)
> mean(user_na_to_na(x1), na.rm = TRUE)
[1] 3
```

## The mixed solution

While solving the problem, this above solution forces two additional operations:

- converting the (already) declared user missing values, and
- employing the `na.rm` argument.

This should not be necessary, especially if (and it is extremely likely that) users may forget the declared missing values are not actually missing values. This scenario is quite possible, many users previously using other software like SPSS or Stata where nothing else should be done after declaring the missing values.

To solve this situation, package **mixed** creates a very similar object, where declared missing values are actually stored (hence interpreted as) regular NA missing values in R.

```
> library(mixed)
> x2 <- mixed_labelled(c(1:5, 99), labels = c(Missing = 99), na_value = 99)
> x2
<mixed_labelled<integer>[6]>
[1]      1      2      3      4      5 NA(99)
Missing values: 99

Labels:
 value   label
    99 Missing
```

It is now obvious that value 99 is not a regular number anymore, but an actual missing value. More importantly, it circumvents the need to convert declared missing values to

regular NAs, since they are already stored as NA values. The average value is calculated simply as:

```
> mean(x2)
[1] 3
```

Notice that neither `user_na_to_na()`, nor employing `na.rm = TRUE` are necessary and, despite being stored as an NA value, the value 99 is not equivalent to an *empty cell*. The information still exists, and it is simply ignored in the calculations.

The `na.rm = TRUE` is only necessary if there are other unexplained missing values in the data:

```
> mean(c(x2, NA))
[1] NA
> mean(c(x2, NA), na.rm = TRUE)
[1] 3
```

As it can be seen, combining on this vector creates a similar one of the same class:

```
> x2 <- c(x2, 97, 99)
> x2
<mixed_labelled<integer>[8]>
[1]      1      2      3      4      5 NA(99)     97 NA(99)
Missing values: 99

Labels:
 value   label
    99 Missing
```

## Similarities and added value

It should be made obvious that packages **haven** and **labelled** are excellent packages which are not inherently doing a bad thing: the very same result is obtained, just via a different route. Package **mixed** should not even be necessary, if the design philosophy of these packages would be different.

The current functions offer an alternative to these packages, with only one but fundamental difference: instead of treating existing values as missing, package **mixed** interprets missing values as existing.

The declared missing values are (just like in package **haven**) identified as NAs, but they can also be compared against the original values:

```
> is.na(x2)
[1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE
```

```
> x2 == 99
[1] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE  TRUE
```

Most functions are designed to be as similar as possible, for instance `value_labels()` to add / change value labels:

```
> value_labels(x2) <- c(DK = 97, NR = 99)
> x2
<mixed_labelled<integer>[8]>
[1]      1      2      3      4      5 NA(99)     97 NA(99)
Missing values: 99

Labels:
 value label
    97    DK
    99    NR
```

The value 97 is now properly labelled, and it can further be declared as missing. Such declarations do not necessarily have to use the main function `mixed_labelled()`, due to the separate functions `missing_values()` and `missing_range()`:

```
> x2 <- c(x2, -3, -1, -2, -5)
> missing_values(x2) <- c(97, 99)
> missing_range(x2) <- c(-1, -5)
> x2
<mixed_labelled<integer>[12]>
 [1]      1      2      3      4      5 NA(99) NA(97) NA(99) NA(-3) NA(-1)
[11] NA(-2) NA(-5)
Missing values: 97, 99
Missing range:  [-5, -1]

Labels:
 value label
    97    DK
    99    NR
```

To ease the translation and inter-operation with packages **haven** and **labelled**, two functions are of interest: `unmix()` and `as_mixed`. The function `unmix()` replaces the NAs with their declared missing values. In addition, the argument `haven` coerces the resulting object to the class `haven_labelled_spss`:

```
> xh <- unmix(x2, haven = TRUE)
```

This object is a perfect equivalent to the one created by package **haven**:

```
> xh
<labelled_spss<integer>[12]>
 [1]  1  2  3  4  5 99 97 99 -3 -1 -2 -5
Missing values: 97, 99
Missing range:  [-5, -1]

Labels:
 value label
    97    DK
    99    NR
```

Naturally, any such object from package **haven** can be coerced to a mixed object:

```
> as_mixed(xh)
<mixed_labelled<integer>[12]>
 [1]      1      2      3      4      5 NA(99) NA(97) NA(99) NA(-3) NA(-1)
[11] NA(-2) NA(-5)
Missing values: 97, 99
Missing range:  [-5, -1]

Labels:
 value label
    97    DK
    99    NR
```

The mixed objects play natively with the base functions `na.omit()` or `na.remove()`, either as standalone vectors or part of a data frame:

```
> dfm <- data.frame(id = sample(1:12, 12), x2)
> dfm
   id     x2
1   8      1
2   7      2
3   5      3
4   2      4
5   9      5
6   6 NA(99)
7   3 NA(97)
8  10 NA(99)
9  12 NA(-3)
10  1 NA(-1)
11  4 NA(-2)
12 11 NA(-5)
```

```
> na.omit(dfm)
  id x2
1  8  1
2  7  2
3  5  3
4  2  4
5  9  5
```

There is an obvious duplication between packages **mixed** and **labelled**. As mentioned, the function `value_labels()` corresponds to the function `val_labels()` in package **labelled**, and the same between functions `variable_labels()` and `var_labels()`.

The list could grow but this is quite unnecessary, as it would be very easy to add methods to the functions from package **labelled**, specific to the objects of class `mixed_labelled` from this package.

One possible example, using the function `to_factor()`. It is currently possible to use it directly over an object from package **mixed**:

```
> x <- mixed_labelled(c(1:4), labels = c(Good = 1, Bad = 5))
> to_factor(unmix(x, haven = TRUE))
[1] Good 2    3    4
Levels: Good 2 3 4 Bad
```

Ideally, these packages should be merged to provide a unified experience, but otherwise they can function either together or separate.