Why the package **declared**

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 all the functionality most users would ever need.

As nice and useful as 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(
    x = c(1:5, -91),
    labels = c(Missing = -91),
    na_value = -91
  )
```

The printed objects from this package nicely display some properties:

```
> x1
<labelled_spss<double>[6]>
[1]   1   2   3   4   5 -91
Missing values: -91

Labels:
 value   label
   -91 Missing
```

There are 5 normal (non-missing) values (supposedly they represent the number of children), and one declared missing value coded -91 This value *acts* as a missing value, but it is different from a regular missing value in R, coded `NA`. The latter 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 -91 is not a valid value either.

It cannot possibly represent -91 children in the household, but for instance it could have meant 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] -12.66667
```

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 declared 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, as many users previously using other software like SPSS or Stata where nothing else should be done after declaring the missing values, may not realize more is needed.

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

```
> library(declared)
> x2 <- declared(
    x = c(1:5, -91),
    labels = c(Missing = -91),
    na_value = -91
  )
```

```
> x2
<declared<integer>[6]>
[1]       1       2       3       4       5 NA(-91)
Missing values: -91

Labels:
 value   label
   -91 Missing
```

The print method makes it obvious the value -91 is not a regular number, but an actual missing value. More importantly, this type of storage circumvents the need to convert user missing values to regular NAs since they are already stored as regular 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, but it is simply ignored in the calculations.

At a first glance, providing a class method for this function seems unnecessary because activating the argument na.rm will return the correct result, anyways. Explaining the importance of the class method requires a discussion about the base R decision to have this argument deactivated by default. This is most likely to alert users about possible problems in the data, since a default value of TRUE would obscure such problems, the mean being calculated irrespective of potentially problematic NA values.

This is where differentiating between empty and declared missing values proves valuable. The declared missing values are neither problematic, nor do they signal potential problems in the data, given that once declaring a reason, it is already known why a particular value is missing.

The truly problematic values are the empty NAs, and the custom class method still allows identifying such values if they exist:

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

Since all declared values are stored as regular NA values, the base function is.na(), as well as all related functions such as anyNA() etc., are unaware and can not differentiate between empty and declared missing values:

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

To overcome this situation, package **declared** complementary provides an additional function to account for the difference:

```
> is.empty(c(x2, NA))
[1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
```

All missing values, empty and declared, play natively with the `NA` oriented, base functions such as `na.omit()` or `na.exclude()`:

```
> na.omit(x2)
<declared<integer>[5]>
[1] 1 2 3 4 5
Missing values: -91

Labels:
 value   label
   -91 Missing
```

It should be made obvious the excellent packages **haven** and **labelled** are not inherently doing a bad thing: the very same result is obtained, just via a different route. Package **declared** was created as an alternative to the design philosophy of these packages, with a fundamental difference: instead of treating existing values as missing, package **declared** interprets missing values as existing.

It does so by storing an additional attribute containing the positions (indexes) of the regular `NA` values in the object which should be treated as missing, and even more so to be interpreted as a particular missing response category, as specified in the value labels attribute.

Similarities and added value

The proposed method to declare missing values is unique in the R ecosystem. Differentiating between empty and declared missing values opens the door to a new set of challenges for which base the R does not have built in functionality.

For instance, the declared missing values can be compared against both the original values and their labels:

```
> x2 == -91
[1] FALSE FALSE FALSE FALSE FALSE  TRUE
> x2 == "Missing"
[1] FALSE FALSE FALSE FALSE FALSE  TRUE
```

Similar methods have been added to the primitive functions "<", ">" and "!=" etc., to allow a fully functional collection of subsetting possibilities.

Combining on this type of vector creates an object of the same class:

```
> x2 <- c(x2, -91)
> x2
<declared<integer>[7]>
[1]       1       2       3       4       5 NA(-91) NA(-91)
Missing values: -91

Labels:
 value   label
   -91 Missing
```

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

```
> value_labels(x2) <- c("Does not know" = -92, "Not responded" = -91)
> x2
<declared<integer>[7]>
[1]       1       2       3       4       5 NA(-91) NA(-91)
Missing values: -91

Labels:
 value         label
   -92 Does not know
   -91 Not responded
```

The value -92 is now properly labelled, and it can further be declared as missing. Such declarations do not necessarily have to use the main function `declared()`, due to the separate functions `missing_values()` and `missing_range()`:

```
> missing_values(x2) <- c(-91, -92)
> missing_range(x2) <- c(-91, -95)
> x2
<declared<integer>[7]>
[1]       1       2       3       4       5 NA(-91) NA(-91)
Missing values: -91, -92
Missing range:  [-95, -91]

Labels:
 value         label
   -92 Does not know
   -91 Not responded
```

To ease the smooth inter-operation with packages **haven** and **labelled**, the following functions are of interest: `undeclare()`, `as.haven()` and `as.declared()`.

The function `undeclare()` replaces the `NA`s with their declared missing values. The result is still an object of class `declared`, but all missing values (and missing range) are stripped off the vector and values are presented as they have been collected. All other attributes of interest (variable and value labels) are retained and printed accordingly.

Activating the argument `drop` eliminates all classes and attributes, returning a regular R object:

```
> undeclare(x2, drop = TRUE)
[1]   1   2   3   4   5 -91 -91
```

The function `as.haven()` coerces the resulting object to the class `haven_labelled_spss`, and the function `as.declared()` reverses the process:

```
> xh <- as.haven(x2)

> xh
<labelled_spss<double>[7]>
[1]   1   2   3   4   5 -91 -91
Missing values: -91, -92
Missing range:  [-95, -91]

Labels:
 value         label
   -92 Does not know
   -91 Not responded

> as.declared(xh)
<declared<integer>[7]>
[1]       1       2       3       4       5 NA(-91) NA(-91)
Missing values: -91, -92
Missing range:  [-95, -91]

Labels:
 value         label
   -92 Does not know
   -91 Not responded
```

The missing values are properly formatted, even inside the base data frame:

```
> dfm <- data.frame(x1 = letters[1:7], x2)

> dfm
  x1      x2
1  a       1
2  b       2
3  c       3
4  d       4
5  e       5
6  f NA(-91)
7  g NA(-91)
```

If users prefer a tibble instead of a data frame, the objects of class `declared` are properly formatted in a similar way to those from package **haven**:

```
> library(tibble)

> as_tibble(dfm)
# A tibble: 7 x 2
  x1                          x2
  <chr>                <int+lbl>
1 a       1
2 b       2
3 c       3
4 d       4
5 e       5
6 f     -91 (NA) [Not responded]
7 g     -91 (NA) [Not responded]
```

Special challenges are associated with sorting and ordering the declared objects, where not all missing values are treated the same.

```
> x3 <- declared(
    x = c(1:5, -91, NA, -92, -91),
    na_value = c(-92, -91)
  )

> sort(x3, na.last = TRUE)
<declared<integer>[9]>
[1]       1       2       3       4       5 NA(-92) NA(-91) NA(-91)      NA
Missing values: -92, -91
```

Sorting in decreasing order applies the same order to the missing values:

```
> sort(x3, na.last = TRUE, decreasing = TRUE)
<declared<integer>[9]>
[1]       5       4       3       2       1 NA(-91) NA(-91) NA(-92)      NA
Missing values: -92, -91
```

This custom function benefits from an additional argument `empty.last` (internally passed to the ordering function), to allow sorting within the missing values:

```
> sort(x3, na.last = TRUE, decreasing = TRUE,
       empty.last = FALSE
  )
<declared<integer>[9]>
[1]       5       4       3       2       1      NA NA(-91) NA(-91) NA(-92)
Missing values: -92, -91
```

All types of variables (categorical and numerical) can have declared missing values. There are always situations values are not missing randomly, but with a specific reason. In social research, respondents often can not, or do not want to provide an answer for a certain question, be it categories of opinions of pure numerical answers like age, income etc.

In base R there is a clear distinction between numerical and categorical variables. For the later, R provides a special type of object called `factor`. The following object simulates such a categorical variable, for instance political orientation:

```
> x4 <- declared(
    x = c(1:3, -91),
    labels = c(Left = 1, Middle = 2, Right = 3, Apolitic = -91),
    na_value = -91,
    label = "Respondent's political orientation"
  )
>
> x4
<declared<integer>[4]> Respondent's political orientation
[1]       1       2       3 NA(-91)
Missing values: -91

Labels:
 value    label
     1     Left
     2   Middle
     3    Right
   -91 Apolitic
```

Such a variable could in principle be constructed directly as a factor:

```
> x5 <- factor(c("Left", "Middle", "Right", "Apolitic"))
> x5
[1] Left     Middle   Right    Apolitic
Levels: Apolitic Left Middle Right
```

The base factor does not provide any possibility to assign specific values for the specific categories, and most importantly does not differentiate between valid values and (declared) missing values. To make the functionality consistent with the base treatment of `NA` values, coercing to factors defaults to dropping the declared missing values:

```
> as.factor(x4)
[1] Left   Middle Right  <NA>
Levels: Left Middle Right
```

To keep the declared missing values as factor levels, the function `as.factor()` has an additional argument called `drop_na` which can be deactivated.

Switching between factors and declared objects, preserving the declared missing values, is straightforward:

```
> as.factor(x4, drop_na = FALSE)
[1] Left     Middle   Right    Apolitic
Levels: Apolitic Left Middle Right

> as.declared(x5, na_values = 1)
<declared<character>[4]>
[1] Left     Middle   Right    Apolitic
Missing values: 1

Labels:
 value    label
     1 Apolitic
     2     Left
     3   Middle
     4    Right
```

When the declared objects are constructed as categorical variables to replace the base factors, an additional function becomes of interest, extracting the categories instead of their values. The function `as.character()` makes this possible, via a dedicated class method for declared objects:

```
> as.character(x4)
[1] "Left"     "Middle"   "Right"    NA
```

The same argument drop_na can be used to preserve the declared missing category labels:

```
> as.character(x4, drop_na = FALSE)
[1] "Left"     "Middle"   "Right"    "Apolitic"
```

And it operates identically if the object was a regular factor:

```
> as.character(as.factor(x4, drop_na = FALSE))
[1] "Left"     "Middle"   "Right"    "Apolitic"
```

It is important to note this argument can only be used when loading the package **declared**. It is not provided in the base package, since these functions are extended by providing class based methods.

To make them resemble more like factors, it is convenient but not at all mandatory to replace the categories with numbers. The declared objects are able to ingest character objects as well as numeric, and the same is valid for the declared missing values:

```
> x6 <- declared(
    x = sample(
      c("Left", "Middle", "Right", "Apolitic"),
      20,
      replace = TRUE
    ),
    na_values = "Apolitic"
  )

> x6
<declared<character>[20]>
 [1]       Left         Right        Right        Middle       Middle
 [6]       Left         Left         Left         Right        Right
[11] NA(Apolitic) NA(Apolitic)       Middle       Right  NA(Apolitic)
[16]      Right  NA(Apolitic)       Right        Right        Right
Missing values: Apolitic
```

Either as character, numeric or categorical, it is possible to declare and use special types of missing values, employing this new object type of class `"declared"`.

Factors and `haven` objects have default coercion methods, but not all types of objects can be automatically coerced to this class. To meet this possibility, the main functions `declared()`, `as.declared()` and `undeclare()` are all generic, allowing full flexibility for any other package to create custom (coercion) methods for different classes of objects, thus facilitating and encouraging a widespread use.

For the remaining examples in this vignette, the following data frame is created for demonstration purposes:

```
> n <- 1234
> set.seed(n)
> DF <- data.frame(
    Area = declared(
      sample(1:2, n, replace = TRUE, prob = c(0.45, 0.55)),
      labels = c(Rural = 1, Urban = 2)
    ),
    Gender = declared(
      sample(1:2, n, replace = TRUE, prob = c(0.55, 0.45)),
      labels = c(Males = 1, Females = 2)
    ),
    Opinion = declared(
      sample(c(1:5, NA, -91), n, replace = TRUE),
      labels = c(
        "Very bad" = 1, "Bad" = 2, "Neither" = 3,
        "Good" = 4, "Very good" = 5, "Don't know" = -91
      ),
      na_values = -91
    ),
    Age = sample(18:90, n, replace = TRUE),
    Children = sample(0:5, n, replace = TRUE)
  )
```

One of the most interesting applications to make use of the declared missing values are the tables of frequencies. The base function `table()` ignores missing values by default, but they can be revealed by using the argument `useNA`:

```
> table(DF$Opinion, useNA = "ifany")

 Very bad       Bad   Neither      Good Very good       <NA>
      180       170       188       171       162       363
```

However, it does not differentiate between empty and declared missing values. Since "Opinion" is the equivalent of a categorical variable, this can be improved through a custom built coercion to the base `factor` class:

```
> table(as.factor(DF$Opinion, drop_na = FALSE), useNA = "ifany")

 Very bad        Bad    Neither       Good  Very good Don't know       <NA>
      180        170        188        171        162        180        183
```

This looks a bit cumbersome, while the dedicated function `w_table()` does the same thing by automatically recognizing objects of class `"declared"`, additionally printing more detailed information:

```
> w_table(DF$Opinion, values = TRUE)

               fre    rel   per   vld   cpd
             ----------------------------
  Very bad     1  180  0.146  14.6  20.7  20.7
       Bad     2  170  0.138  13.8  19.5  40.2
   Neither     3  188  0.152  15.2  21.6  61.8
      Good     4  171  0.139  13.9  19.6  81.4
 Very good     5  162  0.131  13.1  18.6 100.0
         ------
Don't know -91  180  0.146  14.6
            NA  183  0.148  14.8
             ----------------------------
              1234  1.000 100.0
```

The prefix `w_` from the function name stands for "weighted", this being another example of functionality where the declared missing values play a different role than the empty, base `NA` missing values. It is important to differentiate between frequency weights, on one hand, and other probability based, post-stratification weights on one other, the later being thoroughly treated by the specialized package **survey**.

The `w_` family of functions are solely dealing with frequency weights, to allow corrections in descriptive statistics, such as the tables of frequencies and other similar descriptive measures for both categorical and numeric variables.

To exemplify, a frequency weights variable is constructed, to correct for the distributions of gender by males and females, as well as the theoretical distribution by residential areas differentiating between urban and rural settlements.

```
> # Observed proportions
> op <- proportions(with(DF, table(Gender, Area)))

> # Theoretical / population proportions:
> # 53% Rural, and 50% Females
> weights <- rep(c(0.53, 0.47), each=2) * rep(0.5, 4) / op

> DF$fweight <- weights[
    match(10 * DF$Area + DF$Gender, c(11, 12, 21, 22))
  ]
```

The updated frequency table, this time using the frequency weights, can be constructed by passing the weights to the argument `wt`:

```
> with(DF, w_table(Opinion, wt = fweight, values = TRUE))

               fre    rel   per   vld   cpd
             ----------------------------
  Very bad     1  179  0.145  14.5  20.5  20.5
       Bad     2  167  0.135  13.5  19.2  39.7
   Neither     3  187  0.152  15.2  21.4  61.1
      Good     4  171  0.139  13.9  19.6  80.7
 Very good     5  168  0.136  13.6  19.3 100.0
         ------
Don't know -91  179  0.145  14.5
            NA  183  0.148  14.8
             ----------------------------
             1234  1.000 100.0
```

Except for the empty `NA` values, for which the weights cannot be applied, almost all other frequencies (including the one for the declared missing value -91) are now updated by applying the weights. This shows that, despite being interpreted as "missing" values, the declared ones can and should also be weighted, with a very useful result. Other versions of weighted frequencies do exist in R, but a custom one was needed to identify (and weight) the declared missing values.

In the same spirit, many other similar functions are provided such as `w_mean()`, `w_var()`, `w_sd()` etc., and the list will likely grow in the future. They are similar to the base package counterparts, with a single difference: the argument `na.rm` is activated by default, with or without weighting. This is an informed decision about which users are alerted in the functions' respective help pages.

The package **declared** was built with the specific intention to provide a lightweight, zero dependency resource in the R ecosystem. It contains an already extensive, robust and ready to use functionality that duly takes into account the difference between empty and declared missing values.

It extends base R and opens up data analysis possibilities without precedent. By providing generic classes for all its objects and functions, package **declared** is easily extensible to any type of object, for both creation and coercion to class `"declared"`.