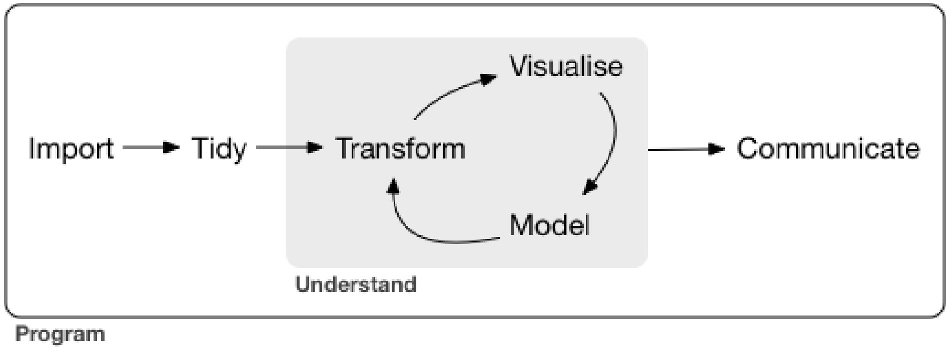


8th Meeting of the Hamburg R-User-Group, 21st Feb 2017

Data Transformation in R
The Tidyverse-Approach of Organizing Data

Dr. Daniel Lüdecke

d.luedecke@uke.de
https://github.com/sjPlot

Universitätsklinikum
Hamburg-Eppendorf

What is the „tidyverse“?

> “The tidyverse is a collection of R packages
> that share common philosophies
> and are designed to work together.”

- This affects *package authors*, when designing packages and think about api‘s…
- … but also *users*, as packages and functions should be designed for humans.

Source: http://tidyverse.org/

What is the „tidyverse“?

“The tidyverse is a collection of R packages
that share common philosophies
and are designed to work together.”

“Design your API primarily so that it is easy to use by humans. Computer efficiency is a secondary concern because the bottleneck in most data analysis is thinking time, not computing time.”

Source: http://tidyverse.org/

two core ideas of the

TIDYVERSE-PHILOSOPHY

Tidyverse-philosophy: two core ideas

- Readable code chunks

  The “pipe”-operator: `%>%`

- Consistent function design

  For instance, data is always the first argument.
  (which derives from the first bullet point)

Readable code chunks: The “pipe”-operator

- Located in the magrittr-package
  (and re-exported by packages like dplyr)
- Aim: to decrease development time and to improve readability and maintainability of code.
- pipe a value or a result forward into an expression or function call:

  `x %>% f1 %>% f2` , rather than `f2(f1(x))`

https://cran.r-project.org/web/packages/magrittr/vignettes/magrittr.html

Readable code chunks: The “pipe”-operator

- Readable code chunks can be considered as “grammar” of coding, which follows the similar intuitive logic from language or thinking

piped code chunk

```
data %>%
  do_first() %>%
  then_second() %>%
  and_then_third() %>%
  finally_last_step()
```

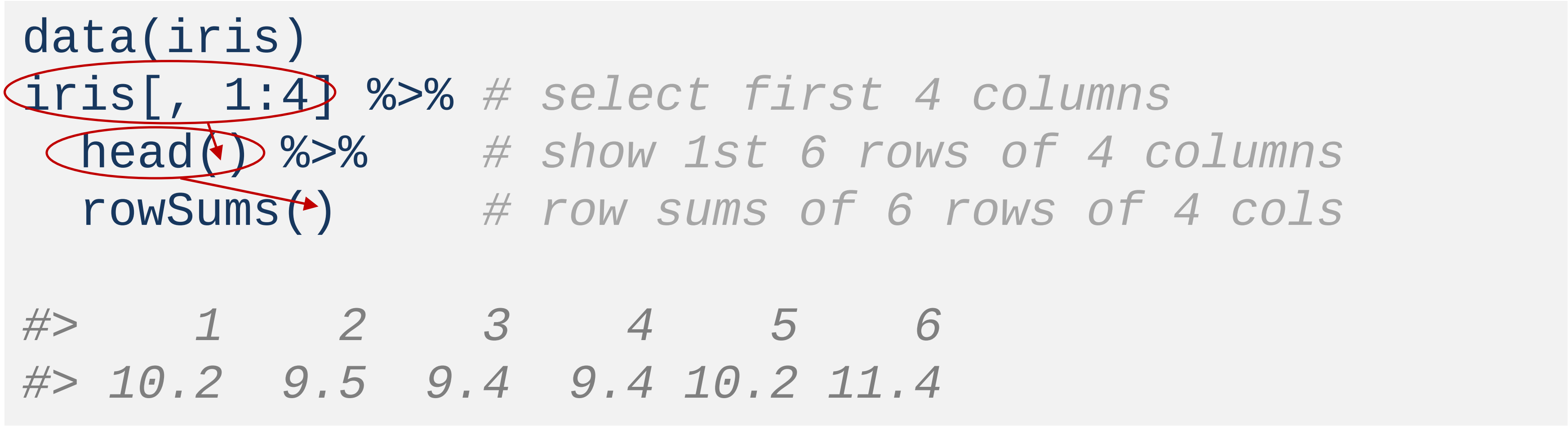
regular code chunk

```
finally_last_step(
  and_then_third(
    then_second(
      do_first(data)
    )
  )
)
```

Readable code chunks: The “pipe”-operator

- Readable code chunks can be considered as “grammar” of coding, which follows the similar intuitive logic from language or thinking

- The pipe-operator takes the output from former function and forwards it *as first argument into the next function*.

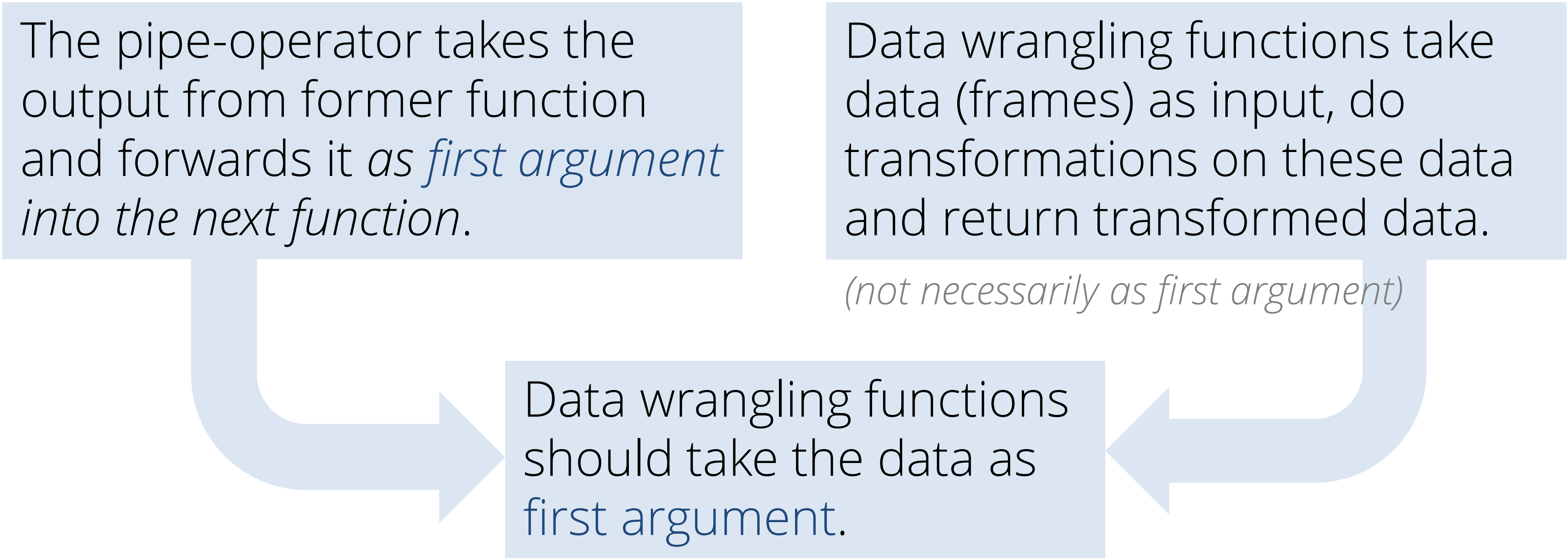


```
data(iris)
iris[, 1:4] %>% # select first 4 columns
  head() %>%    # show 1st 6 rows of 4 columns
  rowSums()     # row sums of 6 rows of 4 cols

#>    1    2    3    4    5    6
#> 10.2  9.5  9.4  9.4 10.2 11.4
```

Readable code chunks: The “pipe”-operator

Idea of pipe-workflow

Idea of data wrangling functions



The pipe-operator takes the output from former function and forwards it *as first argument into the next function*.

Data wrangling functions take data (frames) as input, do transformations on these data and return transformed data.

(not necessarily as first argument)

Data wrangling functions should take the data as first argument.

Consistent function design

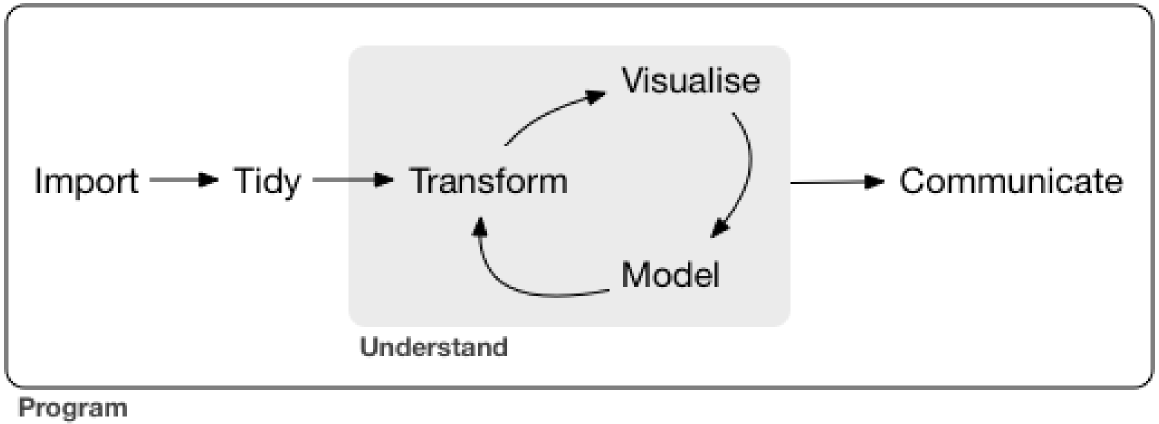
- Data is the first argument.
- „...“-ellipses as second argument.
  - □ works within a pipe-workflow
  - □ allows flexible processing from user-defined amount of variables, expressions etc.

(drawback: all other arguments after "..." need to be explicitly named)

the tidyverse-philosophy and

ORGANIZING DATA

Organizing Data



I. Import of data

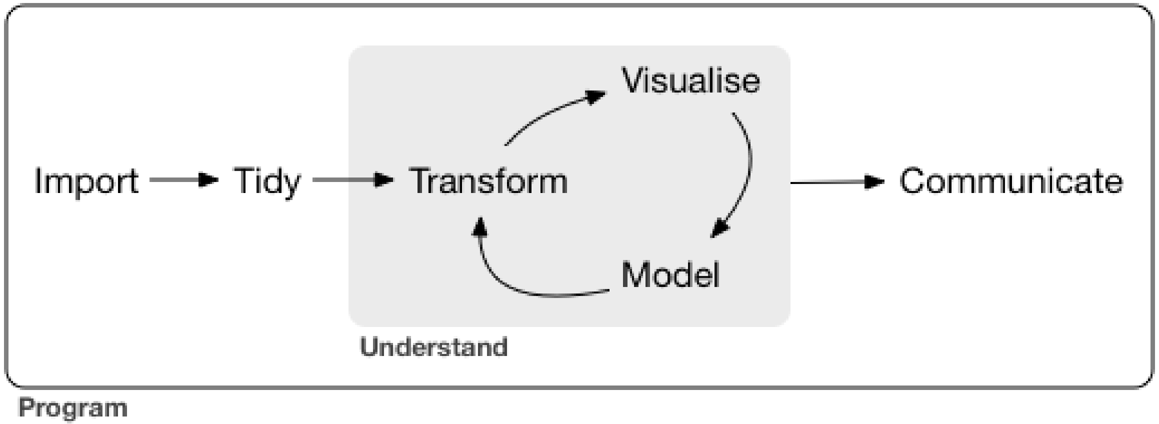
II. First rough tidying

(reshaping, long to wide and vice versa, separating or uniting columns)

III. More specific data tidying and transformation

http://r4ds.had.co.nz/explore-intro.html

Organizing Data



I. Import of data

II. First rough tidying (e.g. package *tidyr*)

(reshaping, long to wide and vice versa, separating or uniting columns)

III. More specific data tidying and transformation (e.g. packages *dplyr* and *sjmisc*)

http://r4ds.had.co.nz/explore-intro.html

Data Tidying and Transformation

WORKING WITH DATA FRAMES

the dplyr-package

Data Transformation in R

- What can we do with *data frames*?

  Examples:

  □ `select()` or `rename()` variables / columns

  □ `filter()` or `slice()` observations

  □ `arrange()` (sort) columns

  □ create new variables with `mutate()`

  □ `summarise()` data

  (typical tasks done with the dplyr-package)

Data Transformation in R

I. From dataset `mtcars`, select variables `mpg` and `gear`

```
data(mtcars)
mtcars %>%
  select(mpg, gear)
```

Data Transformation in R

I. From dataset `mtcars`, select variables `mpg` and `gear`.

II. Only take those cars (*observations*) with mileage of more than 20 miles per gallon.

```
data(mtcars)
mtcars %>%
  select(mpg, gear) %>%
  filter(mpg > 20)
```

Data Transformation in R

I. From dataset `mtcars`, select variables `mpg` and `gear`.

II. Only take those cars (*observations*) with mileage of more than 20 miles per gallon.

III. Group the data by `gear`.

```
data(mtcars)
mtcars %>%
  select(mpg, gear) %>%
  filter(mpg > 20) %>%
  group_by(gear)
```

Data Transformation in R

I. From dataset `mtcars`, select variables `mpg` and `gear`.

II. Only take those cars (*observations*) with mileage of more than 20 miles per gallon.

III. Group the data by `gear`.

IV. Give a summary of how many cars have how many gears.

```
data(mtcars)
mtcars %>%
  select(mpg, gear) %>%
  filter(mpg > 20) %>%
  group_by(gear) %>%
  summarise(n = n())
```

Data Transformation in R

```
data(mtcars)
mtcars %>%
  select(mpg, gear) %>%
  filter(mpg > 20) %>%
  group_by(gear) %>%
  summarise(n = n())

#> # A tibble: 3 × 2
#>     gear     n
#>    <dbl> <int>
#> 1      3     2
#> 2      4    10
#> 3      5     2
```

From all cars with a mileage of > 20 miles per gallon, we have 2 cars with 3 gears, 10 cars with 4 gears and 2 cars with 5 gears.

Data Transformation in R

```
data(mtcars)
mtcars %>%
  select(mpg, gear) %>%
  filter(mpg > 20) %>%
  group_by(gear) %>%
  summarise(n = n())

#> # A tibble: 3 × 2
#>    gear     n
#>   <dbl> <int>
#> 1     3     2
#> 2     4    10
#> 3     5     2
```

```
data(mtcars)
summarise(
  group_by(
    filter(
      select(
        mtcars,
        mpg,
        gear),
      mpg > 20),
    gear),
  n = n())
```

(regular code w/o pipes)

Data Transformation in R

```
data(mtcars)
mtcars %>%
  select(mpg, gear) %>%
  filter(mpg > 20) %>%
  group_by(gear) %>%
  summarise(n = n())
```

```
data(mtcars)
summarise(group_by(filter(select(mtcars, mpg, gear), mpg > 20), gear), n = n())
```

(or as one-liner...)

Data Tidying and Transformation

WORKING WITH VECTORS

the sjmisc-package

Data Transformation in R

- What can we do with *variables*?

  Examples:

  □ `rec()`ode or `dicho()`tomize variables

  □ `std()` (standardize) or `center()` variables

  □ `group_var()`iables

  □ convert variables `to_factor()`, `to_label()`, …

  □ and work with *labelled data*…

  (typical tasks done with the sjmisc-package)

Design of functions in the sjmisc-package

- The returned object for each function equals the type of the data-argument:
  - If the data-argument is a *vector*, the function returns a *vector*.
  - If the data-argument is a *data frame*, the function returns a *data frame*.

Design of functions in the sjmisc-package

```
library(sjmisc)
data(efc)

# returns a vector
x <- rec(efc$e42dep, recodes = "1,2=1; 3,4=2")
str(x)

#>  atomic [1:908] 2 2 2 2 2 2 2 2 2 2 ...
#>  - attr(*, "label")= chr "elder's dependency"
```

Design of functions in the sjmisc-package

```
# returns a data frame (a tibble, to be exactly)
rec(efc, e42dep, recodes = "1,2=1; 3,4=2")

#> # A tibble: 908 × 1
#>    e42dep_r
#>       <dbl>
#> 1         2
#> 2         2
#> 3         2
#> 4         2
#> 5         2
#> 6         2
#> # ... with 902 more rows
```

Data Transformation in R

```
?rec

Usage:

rec(x, ..., recodes, as.num = TRUE, var.label =
    NULL, val.labels = NULL, suffix = "_r")


Arguments:

x     A vector or data frame.
...   Optional, unquoted names of variables. Required, if x
      is a data frame (and no vector) and only selected
      variables from x should be processed. You may also use
      functions like : or dplyr's select_helpers.
recodes   String with recode pairs of old and new values.
```

Data Transformation in R

```
?rec

Usage:

rec(x, ..., recodes, as.num = TRUE, var.label =
    NULL, val.labels = NULL, suffix = "_r")


Arguments:

x    A vector or data frame.
...  Optional, unquoted names of variables. Required, if x
     is a data frame (and no vector) and only selected
     variables from x should be processed. You may also use
     functions like : or dplyr's select_helpers.
recodes   String with recode pairs of old and new values.
```

Data Transformation in R

```
rec(efc, ~contains("cop"), c161sex:c175empl, e42dep,
    recodes = "0,1=0; else=1")
```

Data Transformation in R

```
rec(efc, ~contains("cop"), c161sex:c175empl, e42dep,
    recodes = "0,1=0; else=1")



...   Optional, unquoted names of variables. Required, if x
      is a data frame (and no vector) and only selected
      variables from x should be processed. You may also use
      functions like : or dplyr's select_helpers.
```

Data Transformation in R

```
rec(efc, ~contains("cop"), c161sex:c175empl, e42dep,
    recodes = "0,1=0; else=1")



...   Optional, unquoted names of variables. Required, if x
      is a data frame (and no vector) and only selected
      variables from x should be processed. You may also use
      functions like : or dplyr's select_helpers.


... ⇨ ~contains("cop"), c161sex:c175empl, e42dep
```

Data Transformation in R

```
rec(efc, ~contains("cop"), c161sex:c175empl, e42dep,
    recodes = "0,1=0; else=1")


...   Optional, unquoted names of variables. Required, if x
      is a data frame (and no vector) and only selected
      variables from x should be processed. You may also use
      functions like : or dplyr's select_helpers.

... ⇨ ~contains("cop"), c161sex:c175empl, e42dep
```

- all variables with „cop“ in their name
- all variables from c161sex to c175empl
- and variable e42dep

Data Transformation in R

```
rec(efc, ~contains("cop"), c161sex:c175empl, e42dep,
    recodes = "0,1=0; else=1")

#>  # A tibble: 908 × 13
#>    c161sex_r c172code_r c175empl_r e42dep_r c82cop1_r
#>        <dbl>      <dbl>      <dbl>    <dbl>     <dbl>
#> 1          1          1          0        1         1
#> 2          1          1          0        1         1
#> 3          0          0          0        1         1
#> 4          0          1          0        1         1
#> 5          1          1          0        1         1
#> 6          0          1          0        1         1
#> 7          1          1          0        1         1
#> 8          1          1          0        1         1
#> 9          1         NA          0        1         1
#> ... with 898 more rows, and 8 more variables: c86cop5_r
<dbl>, c87cop6_r <dbl>,...
```

The Best of Both Worlds

INTEGRATING SJMISC AND DPLYR

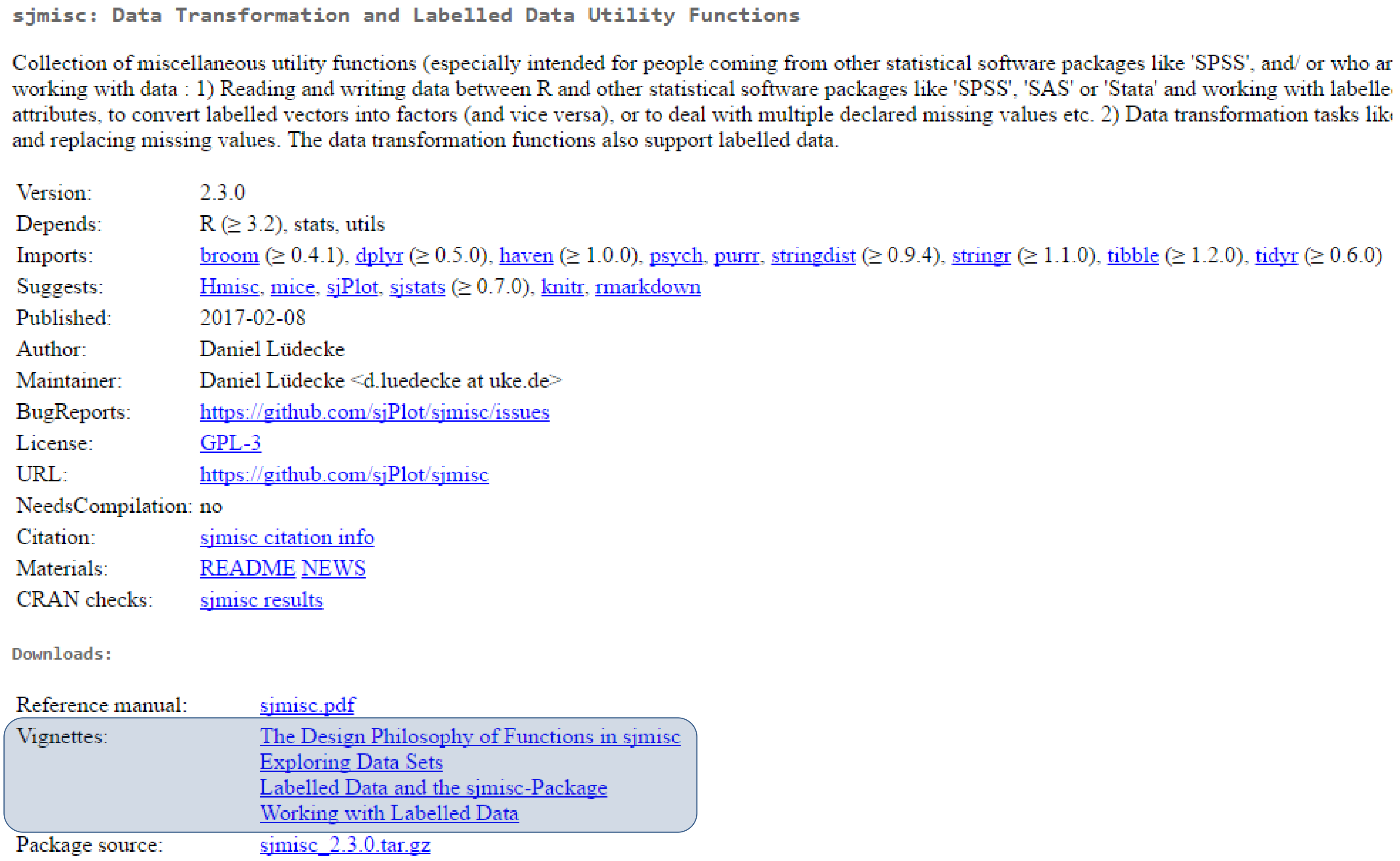
Combining sjmisc and dplyr

```
efc %>%
  select(c82cop1, c83cop2) %>%
  rec(recodes = "1,2=0; 3:4=2")

#> # A tibble: 908 × 2
#>     c82cop1_r c83cop2_r
#>         <dbl>     <dbl>
#> 1           2         0
#> 2           2         2
#> 3           0         0
#> 4           2         0
#> 5           2         0
#> 6           0         0
#> 7           2         0
#> 8           2         0
#> # ... with 900 more rows
```

Combining sjmisc and dplyr

```
efc %>%
  select(c82cop1, c83cop2) %>%
  mutate(
    c82cop1_dicho = rec(c82cop1, recodes = "1,2=0; 3:4=2"),
    c83cop2_dicho = rec(c83cop2, recodes = "1,2=0; 3:4=2")
  ) %>%
  head()

#>   c82cop1 c83cop2 c82cop1_dicho c83cop2_dicho
#> 1       3       2             2             0
#> 2       3       3             2             2
#> 3       2       2             0             0
#> 4       4       1             2             0
#> 5       3       2             2             0
#> 6       2       2             0             0
```

CRAN - Package sjmisc



sjmisc: Data Transformation and Labelled Data Utility Functions

Collection of miscellaneous utility functions (especially intended for people coming from other statistical software packages like 'SPSS', and/ or who ar working with data : 1) Reading and writing data between R and other statistical software packages like 'SPSS', 'SAS' or 'Stata' and working with labelle attributes, to convert labelled vectors into factors (and vice versa), or to deal with multiple declared missing values etc. 2) Data transformation tasks lik and replacing missing values. The data transformation functions also support labelled data.

Version:	2.3.0
Depends:	R (≥ 3.2), stats, utils
Imports:	broom (≥ 0.4.1), dplyr (≥ 0.5.0), haven (≥ 1.0.0), psych, purrr, stringdist (≥ 0.9.4), stringr (≥ 1.1.0), tibble (≥ 1.2.0), tidyr (≥ 0.6.0)
Suggests:	Hmisc, mice, sjPlot, sjstats (≥ 0.7.0), knitr, rmarkdown
Published:	2017-02-08
Author:	Daniel Lüdecke
Maintainer:	Daniel Lüdecke <d.luedecke at uke.de>
BugReports:	https://github.com/sjPlot/sjmisc/issues
License:	GPL-3
URL:	https://github.com/sjPlot/sjmisc
NeedsCompilation:	no
Citation:	sjmisc citation info
Materials:	README NEWS
CRAN checks:	sjmisc results

Downloads:

Reference manual:	sjmisc.pdf
Vignettes:	The Design Philosophy of Functions in sjmisc Exploring Data Sets Labelled Data and the sjmisc-Package Working with Labelled Data
Package source:	sjmisc_2.3.0.tar.gz

Thanks for your attention!

Dr. Daniel Lüdecke

Universitätsklinikum Hamburg-Eppendorf
Institut für Medizinische Soziologie
Martinistraße 52, W37/8. Stock
D-20246 Hamburg

d.luedecke@uke.de
https://github.com/sjPlot