# Use cttab in the Analysis

Alimu Dayimu

2022-04-04

## What this Document Does

The package `cctu` has been updated to utilize the DLU file and CLU file to facilitate the analyzing process. The newly added functions will not affect previous process. But here, in this document we are going to learn some new functions provided in the package to facilitate your analysis. Before you start, you should get yourself familiarized with the `cctu` package by looking through the `Analysis Template` vignette.

## What's new

### Variable and value label

If you are familiar with `SAS`, `Stata` or `SPSS`, you should already know there's a variable label and value label (variable format in some). The variable label gives you the description of the variable, and the value label is to explain what the values stand for. The variable will stay as a numeric but has categories attached to it. Which means, you can subset or manipulate it like a numerical variable but report the data as a categorical one. It is important to understand this, let's use `mtcars` dataset and demonstrate how it works.

```
data(mtcars)
#> Warning in find.package(package, lib.loc, verbose = verbose): package 'cctu' found more than once, u
#>   "C:/Users/sjb277/AppData/Local/Temp/RtmpOCZRzz/temp_libpath872065172a7c/cctu",
#>   "U:/My Documents/R/win-library/4.1/cctu"
#> Warning in find.package(package, lib.loc, verbose = verbose): package 'base' found more than once, u
#>   "C:/PROGRA~1/R/R-41~1.1/library/base",
#>   "C:/Program Files/R/R-4.1.1/library/base"
#> name=mtcars:  NOT found in names() of Rdata.rds, i.e.,
#>  billboard,construction,fish_encounters,population,relig_income,smiths,table1,table2,table3,table4a,
#> name=mtcars:  NOT found in names() of Rdata.rds, i.e.,
#>  diamonds,economics,economics_long,faithfuld,luv_colours,midwest,mpg,msleep,presidential,seals,txhou
#> name=mtcars:  NOT found in names() of Rdata.rds, i.e.,
#>  band_instruments,band_instruments2,band_members,starwars,storms
#> name=mtcars:  NOT found in names() of Rdata.rds, i.e.,
#>  meta_table_example
#> name=mtcars:  found in Rdata.rds
# Assign variable label
var_lab(mtcars$am) <- "Transmission"

# Assign value label with named vector
val_lab(mtcars$am) <- c("Automatic" = 0, "Manual"=1)
str(mtcars$am)
#>  num [1:32] 1 1 1 0 0 0 0 0 0 0 ...
#>  - attr(*, "label")= chr "Transmission"
#>  - attr(*, "labels")= Named num [1:2] 0 1
```

```
#>   ..- attr(*, "names")= chr [1:2] "Automatic" "Manual"
```

As you can above, `label` and `labels` attributes were added. But the variable is numeric. You can still summarize it as numeric variable as below:

```
summary(mtcars$am)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>  0.0000  0.0000  0.0000  0.4062  1.0000  1.0000
```

You can convert the value label to factor just before your final analysis. The `to_factor` function replace the value with labels and convert the variable to factor.

```
# Extract variable label
var_lab(mtcars$am)
#> [1] "Transmission"
# Convert variable to factor with labels attached to it
table(to_factor(mtcars$am))
#>
#> Automatic    Manual
#>        19        13
```

But be cautious, **some R process might drop the variable or value label attributes**. Normally it won't, but you should check it before report if you are not sure. For example, `as.numeric`, `as.character` and `as.logiac` will drop the variable and value label. There are `to_numeric`, `to_character` and `to_logical` can be used to convert the data type. You can also use the `copy_lab` function to copy the variable and value labels from the other variable.

You can use `var_lab` to extract the variable label or assign one. And use `has.label` to check if the variable has any variable label with it. You may also want to use `drop_lab` to drop the variable label.

For value label, you can use `val_lab` to extract or assign value label. And use `has.labels` to check if the variable has a value label. There are also `unval` function to drop the value label and `lab2val` to replace the `data.frame` value to its corresponding value labels.

## MACRO dataset utility functions

There are new functions have been added to utilize the DLU and CLU files for the data analysis. The `apply_lus` function uses DLU and CLU file to assign variable and value labels to the dataset. This will also convert the dataset based on the variable type as in the DLU file. The `extract_form` can extract MACRO data by form and visits. The data will be converted to `data.table` class, which is an extension of the `data.frame` and works exactly the same. It is a fantastic package with lots of data manipulation capability, you should seek the website for more details. But one thing to remember is that all the names in the variable selection will be considered as a variable of the data.

```
vars <- c("mpg", "am")
# You can do this in the normal data.frame
mtcars[,vars]

# But you can't do this for the data.table
dat <- data.table::data.table(mtcars)
mtcars[,vars]

# You need to add with=FALSE to do that
mtcars[,vars, with=FALSE]
```

### Table populate function

You might already knew how to use the `sumby` function, but now a new function called `cttab` has been added. You can feed the labelled data to this function and it will populate a summary table. Report data by treatment group, stratify tables by visit. Also, you can report variable based on some conditions and group the variable in the report. It generates a missing report internally and you can dump the missing report at the end. No extra step is needed. The remaining of this document will show you with a working case

## Working example

In this section, we will show how to populate tables.

### Data reading

You should read the data as before, but you can and should read the data by setting all the columns to character. This will be handled later.

```
# Read example data
dt <- read.csv(system.file("extdata", "pilotdata.csv", package="cctu"),
               colClasses = "character")

# Read DLU and CLU
dlu <- read.csv(system.file("extdata", "pilotdata_dlu.csv", package="cctu"))
clu <- read.csv(system.file("extdata", "pilotdata_clu.csv", package="cctu"))
```

If you have multiple datasets, you should combine them at this stage. Next, we will apply the DLU and CLU files to the dataset.

```
# Create subjid
dt$subjid <- substr(dt$USUBJID, 8, 11)

# Apply CLU and DLU files
dt <- apply_lus(dt, dlu = dlu, clu = clu)

# Give new variable a label
var_lab(dt$subjid) <- "Subject ID"
```

After this, you should follow the `Analysis Template` vignette and setup the population etc. For example:

```
set_meta_table(cctu::meta_table_example)
#Create the population table
popn <- dt[,"subjid", drop=FALSE]
popn$safety <- TRUE

create_popn_envir("dt", popn)
.reserved <- ls()
```

Next we will do some data analysis.

### Data analysis

After you have attached the population, next thing you may want to do is extract a particular form from the data. For the table, assume we are reporting age, sex and BMI from the patient registration form by treatment arm. For demonstrating the filtering, we only report non-white patients' BMI.

```
# Attach population
attach_pop("1.1")
```

```r
# Extract patient patient registration form and keep subjid variable
df <- extract_form(dt, "PatientReg", vars_keep = c("subjid"))

# Now report Age, Sex and BMI. For BMI, report not white only
X <- cttab(vars = c("AGE", "SEX", "BMIBL"), # Variable to report
           group = "ARM",                   # Group variable
           data = df,                       # Data
           select = c("BMIBL" = "RACEN != 1")) # Filter for variable BMI

# Write table
write_table(X)
```

### Missing data report

The `cttab` function will report the missing internally. You can use the following to get the missing report.

```r
# This will save the missing report under Output folder
# Or you can set the output folder and name
dump_missing_report()

# Pull out the missing report if you want
miss_rep <- get_missing_report()

# Reset missing report
reset_missing_report()
```

After this, you can finish the remaining as in the `Analysis Template` vignette.

## More to cttab

As you have seen previously, the `cttab` function can easily populate simple tables.

### Simple table

Table only some variables, no treatment arm or variable selection.

```r
X <- cttab(vars = c("AGE", "SEX", "BMIBL"), # Variable to report
           data = df,                       # Data
```

### By group and filter

This is what we have seen before

```r
X <- cttab(vars = c("AGE", "SEX", "BMIBL"), # Variable to report
           group = "ARM",                   # Group variable
           data = df,                       # Data
           select = c("BMIBL" = "RACEN != 1")) # Filter for variable BMI
```

### Split table row by visit

You can define `row_split` parameter to the name of visit or repeat variable.

```r
attach_pop("1.1")
df <- extract_form(dt, "Lab", vars_keep = c("subjid", "ARM"))
```

```
X <- cttab(vars = c("AST", "BILI", "ALT"),
           group = "ARM",
           data = df,
           row_split = "AVISIT",            # Visit variable
           select = c("ALT" = "PERF == 1"))
```

## Group variable

In this example, we will report demographic variable, lab results and lab abnormality. Variables will be grouped, no group name will be given to demographic variables, "Blood" to lab results and "Pts with Abnormal" to lab abnormality. Here, we count the number of patients with abnormal lab results. The `cttab` will report the count and percentage of `TRUE`. This is useful if you want to report patient numbers for different condition that belong to one category. Below is how to do it:

```
# Prepare data as before
attach_pop("1.1")
df <- extract_form(dt, "PatientReg", vars_keep = c("subjid"))
base_lab <- extract_form(dt, "Lab", visit = "SCREENING",
                         vars_keep = c("subjid"))

# Define abnormal
base_lab$ABNORMALT <- base_lab$ALT > 22.5
var_lab(base_lab$ABNORMALT) <- "ALT abnormal"
base_lab$ABNORMAST <- base_lab$AST > 25.5
var_lab(base_lab$ABNORMAST) <- "AST abnormal"

df <- merge(df, base_lab, by = "subjid")

# Table
X <- cttab(vars = list(c("AGE", "SEX", "BMIBL"),
                       # Group lab variable
                       "Blood" = c("ALT", "AST"),
                       # Group abnormal variable
                       "Pts with Abnormal" = c("ABNORMAST", "ABNORMALT")),
           group = "ARM",
           data = df,
           # Add some filtering
           select = c("BMIBL" = "RACEN != 1",
                      "ALT" = "PERF == 1"))
```

## Rounding

The default behaviour of this function is to keep one digits for the percentage and 3 significant value for the numerical values. The default rounding function is `signif_pad`, you can also use `round` or `round_pad` to keep digits in the summary. The `format_percent` function is used to format the percentage values. There is `format_pval` might be useful to you.

```
X <- cttab(vars = c("AGE", "SEX", "BMIBL"), # Variable to report
           group = "ARM",                   # Group variable
           data = df,                       # Data
           digits = 2,                      # Keep 2 digits for numerical
           digits_pct = 1,                  # Keep 1 digits for percentage
           rounding_fn = round)             # Use function round for rounding
```

## One more thing

Tables can be cross-referenced in the word. Don't forgot to try writing narratives by cross referencing tables. Have fun!