Package ‘cchsflow’

March 30, 2020

Type Package

Title Transforming and Harmonizing CCHS Variables

Version 1.6.0

Date 2020-03-30

Depends R (>= 3.2), haven (>= 1.1.2), dplyr (>= 0.8.2), sjlabelled (>= 1.0.17), stringr (>= 1.2.0), magrittr

Description Supporting the use of the Canadian Community Health Survey (CCHS) by transforming variables from each cycle into harmonized, consistent versions that span survey cycles (currently, 2001 to 2014). CCHS data used in this library is accessed and adapted in accordance to the Statistics Canada Open Licence Agreement. This package uses rec_with_table(), which was developed from 'sjmisc' rec(). Lüdecke D (2018). ``sjmisc: Data and Variable Transformation Functions''. Journal of Open Source Software, 3(26), 754. <doi:10.21105/joss.00754>.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

URL `https://github.com/Big-Life-Lab/cchsflow`

BugReports `https://github.com/Big-Life-Lab/cchsflow/issues`

RoxygenNote 7.1.0

Suggests testthat (>= 2.1.0)

NeedsCompilation no

Author Doug Manuel [aut, cph] (<https://orcid.org/0000-0003-0912-0845>),
Warsame Yusuf [aut, cre],
Rostyslav Vyuha [aut],
Carol Bennett [aut],
Yulric Sequeira [ctb],
The Ottawa Hospital [cph]

Maintainer Warsame Yusuf <`waryusuf@ohri.ca`>

Repository CRAN

Date/Publication 2020-03-30 17:10:02 UTC

R topics documented:

`adl_fun` *Derived needs help with tasks*

Description

This derived variable (ADL_der) is based on the CCHS derived variable ADLF6R which flags respondents who need help with tasks based on their response to the various activities of daily living (ADL) variables.

Usage

```
adl_fun(ADL_01, ADL_02, ADL_03, ADL_04, ADL_05)
```

Arguments

`ADL_01`	Needs help preparing meals
`ADL_02`	Needs help getting to appointments/errands
`ADL_03`	Needs help doing housework
`ADL_04`	Needs help doing personal care
`ADL_05`	Needs help moving inside house

Details

The CCHS derived variable ADLF6R uses different ADL variables across the various CCHS survey cycles. This newly derived variable (ADL_der) uses ADL variables that are consistent across CCHS cycles.

In the 2001 CCHS survey cycle, the ADLF6R variable examines the following ADL variables:

1. ADL_01 - Needs help preparing meals
2. ADL_02 - Needs help getting to appointments/errands
3. ADL_03 - Needs help doing housework
4. ADL_04 - Needs help doing personal care
5. ADL_05 - Needs help moving inside house
6. ADL_07 - Needs help doing heavy household chores

In the 2003-2005 CCHS survey cycles, the ADLF6R variable examines the following ADL variables:

1. ADL_01 - Needs help preparing meals
2. ADL_02 - Needs help getting to appointments/errands
3. ADL_03 - Needs help doing housework
4. ADL_04 - Needs help doing personal care
5. ADL_05 - Needs help moving inside house
6. ADL_06 - Needs help doing finances

7. ADL_07 - Needs help doing heavy household chores

In the 2007-2014 CCHS survey cycles, the ADLF6R variable examines the following ADL variables:

1. ADL_01 - Needs help preparing meals
2. ADL_02 - Needs help getting to appointments/errands
3. ADL_03 - Needs help doing housework
4. ADL_04 - Needs help doing personal care
5. ADL_05 - Needs help moving inside house
6. ADL_06 - Needs help doing finances

This newly derived variable (ADL_der) uses ADL_01 to ADL_05 which are consistent across all survey cycles. For any single CCHS survey year, it is appropriate to use ADLF6R. ADL_der is recommended when using multiple survey cycles.

Value

A derived variable (ADL_der) with 2 categories:

1. - Needs help with tasks
2. - Does not need help with tasks

Examples

```
# Using adl_fun() to create ADL_der values across CCHS cycles
# adl_fun() is specified in variable_details.csv along with the
# CCHS variables and cycles included.

# To transform ADL_der, use rec_with_table() for each CCHS cycle
# and specify ADL_der, along with the various ADL variables.
# Then by using bind_rows() you can combine ADL_der across cycles.

library(cchsflow)
adl2001 <- rec_with_table(
  cchs2001_p, c(
    "ADL_01", "ADL_02", "ADL_03", "ADL_04", "ADL_05", "ADL_der"
  )
)

head(adl2001)

adl2009_2010 <- rec_with_table(
  cchs2009_2010_p, c(
    "ADL_01", "ADL_02", "ADL_03", "ADL_04", "ADL_05", "ADL_der"
  )
)

tail(adl2009_2010)

combined_adl <- bind_rows(adl2001, adl2009_2010)
```

```
head(combined_adl)

tail(combined_adl)

# Using adl_fun() to generate to ADL_der based on user inputted values.
#
# Let's say you do not need help preparing meals, you need help getting to
# appointments or errands, you need help doing housework, do not need help
# doing personal care, and do not need help moving inside the house. Using
# adl_fun() we can check if you need help doing tasks

ADL_der <- adl_fun(2, 1, 1, 2, 2)

print(ADL_der)
```

age_cat_fun *Derived categorical age*

Description

This is a derived categorical age variable (DHHGAGE_C) that groups various age categories across all CCHS cycles. This is based on the continuous age variable (DHHGAGE_cont) that is harmonious across all CCHS cycles.

The categories of this new age variable are as follows:

1. 12 to 14 years
2. 15 to 17 years
3. 18 to 19 years
4. 20 to 24 years
5. 25 to 29 years
6. 30 to 34 years
7. 35 to 39 years
8. 40 to 44 years
9. 45 to 49 years
10. 50 to 54 years
11. 55 to 59 years
12. 60 to 64 years
13. 65 to 69 years
14. 70 to 74 years
15. 75 to 79 years
16. 80 years or more

Usage

```
age_cat_fun(DHHGAGE_cont)
```

Arguments

`DHHGAGE_cont`	continuous age variable

Details

The categories in the grouped age variable (DHHGAGE) vary between CCHS cycles. As such, a continuous age variable (DHHGAGE_cont) was created that harmonized age across all CCHS cycles by taking the midpoint of each age category. This new age variable (DHHGAGE_C) categorizes age based on the categories used in CCHS cycles from 2007 to 2014.

Value

a categorical age variable (DHHGAGE_C)

Examples

```
# Using age_cat_fun() to create categorical age values from DHHGAGE_cont
# age_cat_fun() is specified in variable_details.csv along with the CCHS
# variables and cycles included.

# To generate DHHGAGE_C in a cycle, use rec_with_table() and specify
# DHHGAGE_C along with DHHGAGE_cont.

library(cchsflow)

cat_age2009_2010 <- rec_with_table(
   cchs2009_2010_p,  c(
     "DHHGAGE_cont", "DHHGAGE_C"
     )
   )
```

`ALCDTTM` *Type of drinker (12 months)*

Description

NOTE: this is not a function.

This is a categorical variable derived by Statistics Canada that uses various intermediate alcohol variables to categorize individuals into 3 distinct groups:

1. Regular Drinker
2. Occasional Drinker
3. No drink in the last 12 months.

Usage

```
ALCDTTM(ALCDTTM)
```

Arguments

`ALCDTTM`	cchsflow variable name for type of drinker (12 months)

Details

This variable was introduced in the 2007-2008 cycle of the CCHS, and became the sole derived variable that categorized people into various drinker types from 2009 onwards. Unlike ALCDTYP, this variable does not distinguish between former and never drinkers.

Examples

```
library(cchsflow)
 ?ALCDTTM
```

`ALCDTYP` *Type of drinker*

Description

NOTE: this is not a function.

This is a categorical variable derived by Statistics Canada that uses various intermediate alcohol variables to categorize individuals into 4 distinct groups:

1. Regular Drinker
2. Occasional Drinker
3. Former Drinker
4. Never Drinker

Usage

```
ALCDTYP(ALCDTYP)
```

Arguments

`ALCDTYP`	cchsflow variable name for type of drinker

Details

This variable is used in CCHS cycles from 2001 to 2007. How it was derived remained consistent during these years.

Starting in 2007, Statistics Canada created a derived variable that looked at drinking type in the last 12 months. This new derived variable did not distinguish between former and never drinkers. If your research requires you to differentiate between former and never drinkers, we recommend using earlier cycles of the CCHS.

Examples

```
library(cchsflow)
 ?ALCDTYP
```

ALWDDLY *Average daily alcohol consumption*

Description

NOTE: this is not a function.

This is a continuous variable derived by Statistics Canada that quantifies the mean daily consumption of alcohol. This takes the value of ALWDWKY and divides it by 7.

Usage

```
ALWDDLY(ALWDDLY)
```

Arguments

ALWDDLY	cchsflow variable name for average daily alcohol consumption

Details

This variable is present in every CCHS cycle used in cchsflow, and how it was derived remains consistent.

Examples

```
library(cchsflow)
 ?ALWDDLY
```

ALWDWKY *Number of drinks consumed in the past week*

Description

NOTE: this is not a function.

This is a continuous variable derived by Statistics Canada that quantifies the amount of alcohol that is consumed in a week. This is calculated by adding the number of drinks consumed each day in the past week. Respondents of each CCHS cycle are asked how much alcohol they have consumed each day in the past week (ie. how much alcohol did you consume on Sunday, how much did you consume on Monday etc.). Each day is considered an individual variable and ALWDWKY takes the sum of all daily variables.

Usage

```
ALWDWKY(ALWDWKY)
```

Arguments

`ALWDWKY`	cchsflow variable name for number of drinks consumed in the past week

Details

This variable is present in every CCHS cycle used in cchsflow, and how it was derived remains consistent.

Examples

```
library(cchsflow)
 ?ALWDWKY
```

`binge_drinker_fun` *Binge drinking*

Description

This function creates a derived categorical variable that flags for binge drinking based on the number drinks consumed on a single day.

Usage

```
binge_drinker_fun(
  DHH_SEX,
  ALW_1,
  ALW_2A1,
  ALW_2A2,
  ALW_2A3,
  ALW_2A4,
  ALW_2A5,
  ALW_2A6,
  ALW_2A7
)
```

Arguments

`DHH_SEX`	sex of respondent (1 - male, 2 - female)
`ALW_1`	Drinks in the last week (1 - yes, 2 - no)
`ALW_2A1`	Number of drinks on Sunday
`ALW_2A2`	Number of drinks on Monday

`ALW_2A3`	Number of drinks on Tuesday
`ALW_2A4`	Number of drinks on Wednesday
`ALW_2A5`	Number of drinks on Thursday
`ALW_2A6`	Number of drinks on Friday
`ALW_2A7`	Number of drinks on Saturday

Details

In health research, binge drinking is defined as having an excess amount of alcohol in a single day. For males, this is defined as having five or more drinks; and for females it is four or more drinks. In the CCHS, respondents are asked to count the number of drinks they had during each day of the last week.

Value

Categorical variable (binge_drinker) with two categories:

1. 1 - binge drinker
2. 2 - non-binge drinker

Examples

```
# Using binge_drinker_fun() to create binge_drinker values across CCHS cycles
# binge_drinker_fun() is specified in variable_details.csv along with the
# CCHS variables and cycles included.

# To transform binge_drinker, use rec_with_table() for each CCHS cycle
# and specify binge_drinker, along with the various alcohol and sex
# variables. Then by using bind_rows() you can combine binge_drinker
# across cycles.

library(cchsflow)
binge2001 <- rec_with_table(
  cchs2001_p, c(
    "ALW_1", "DHH_SEX", "ALW_2A1", "ALW_2A2", "ALW_2A3", "ALW_2A4",
    "ALW_2A5", "ALW_2A6", "ALW_2A7", "binge_drinker"
  )
)

head(binge2001)

binge2009_2010 <- rec_with_table(
  cchs2009_2010_p, c(
    "ALW_1", "DHH_SEX", "ALW_2A1", "ALW_2A2", "ALW_2A3", "ALW_2A4",
    "ALW_2A5", "ALW_2A6", "ALW_2A7", "binge_drinker"
  )
)

tail(binge2009_2010)
```

```
combined_binge <- bind_rows(binge2001, binge2009_2010)

head(combined_binge)

tail(combined_binge)

# Using binge_drinker_fun() to generate binge_drinker with user-inputted
# values.
#
# Let's say you are a male, and you had drinks in the last week. Let's say
# you had 3 drinks on Sunday, 1 drink on
# Monday, 6 drinks on Tuesday, 0 drinks on Wednesday, 3 drinks on Thurday,
# 8 drinks on Friday, and 2 drinks on Saturday. Using binge_drinker_fun(),
# we can check if you would be classified as a drinker.

binge <- binge_drinker_fun(DHH_SEX = 1, ALW_1 = 1, ALW_2A1 = 3, ALW_2A2 = 1,
                           ALW_2A3 = 6, ALW_2A4 = 0, ALW_2A5 = 3,
                           ALW_2A6 = 8, ALW_2A7 = 2)

print(binge)
```

`bmi_fun` *Body Mass Index (BMI) derived variable*

Description

This function creates a harmonized BMI variable. The BMI variable provided by the CCHS calculates BMI using methods that vary across cycles, leading to measurement error when using multiple CCHS cycles. In certain CCHS cycles (2001-2003, 2007+), there are age restrictions in which respondents under the age of 20 and over the age of 64 were not included. Across all CCHS cycles, female respondents who identified as being pregnant were excluded; and in certain CCHS cycles (2003-2007, 2013-2014), females who did not answer the pregnancy question were coded as NS (not stated) for HWTGBMI. As well, in certain CCHS cycles (2001-2003, 2009-2014), respondents outside certain height and weight ranges (0.914-2.108m for height, 0-260kg for weight) were excluded from HWTGBMI.

bmi_fun() creates a derived variable (HWTGBMI_der) that is harmonized across all CCHS cycles. This function divides weight by the square of height.

Usage

```
bmi_fun(HWTGHTM, HWTGWTK)
```

Arguments

`HWTGHTM`	CCHS variable for height (in meters)
`HWTGWTK`	CCHS variable for weight (in kilograms)

Details

For HWTGBMI_der, there are no restrictions to age, height, weight, or pregnancy status. While pregnancy was consistent across all CCHS cycles, its variable (MAM_037) was not available in the PUMF CCHS datasets so it could not be harmonized and included into the function.

For any single CCHS survey year, it is appropriate to use the CCHS BMI variable (HWTGBMI) that is also available on cchsflow. HWTGBMI_der is recommended when using multiple survey cycles.

HWTGBMI_der uses the CCHS variables for height and weight that have been transformed by cchsflow. In order to generate a value for BMI across CCHS cycles, height and weight must be transformed and harmonized.

Value

numeric value for BMI in the HWTGBMI_der variable

Note

In earlier CCHS cycles (2001 and 2003), height was collected in inches; while in later CCHS cycles (2005+) it was collected in meters. To harmonize values across cycles, height was converted to meters (to 3 decimal points). Weight was collected in kilograms across all CCHS cycles, so no transformations were required in the harmonization process.

Examples

```
# Using bmi_fun() to create BMI values between cycles
# bmi_fun() is specified in variable_details.csv along with the
# CCHS variables and cycles included.

# To transform the derived BMI variable, use rec_with_table() for each cycle
# and specify HWTGBMI_der, along with height (HWTGHTM) and weight (HWTGWTK).
# Then by using dplyr::bind_rows(), you can combined HWTGBMI_der across
# cycles.

library(cchsflow)
bmi2001 <- rec_with_table(
  cchs2001_p, c(
    "HWTGHTM",
    "HWTGWTK", "HWTGBMI_der"
  )
)

head(bmi2001)

bmi2011_2012 <- rec_with_table(
  cchs2011_2012_p, c(
    "HWTGHTM",
    "HWTGWTK", "HWTGBMI_der"
  )
)
```

```
tail(bmi2011_2012)

combined_bmi <- bind_rows(bmi2001, bmi2011_2012)
head(combined_bmi)
tail(combined_bmi)

# Using bmi_fun() to generate a BMI value with user inputted height and
# weight values. bmi_fun() can also generate a value for BMI if you input a
# value for height and weight. Let's say your height is 170cm (1.7m) and
# your weight is 50kg, your BMI can be calculated as follows:

library(cchsflow)
BMI <- bmi_fun(HWTGHTM = 1.7, HWTGWTK = 50)
print(BMI)
```

---

cchs2001_p *2001 CCHS PUMF subset data (200 respondents)*

Description

This is a subset of 200 observations from the 2001 cycle of the Canadian Community Health Survey (CCHS) Public Use Microdata file (PUMF) dataset. The CCHS survey is conducted by Statistics Canada.

Details

See here for the open license. Source from Statistics Canada, Canadian Community Health Survey PUMF, accessed Jan 2020. Reproduced and distributed on an "as is" basis with the permission of Statistics Canada.

Long name: cchs-82M0013-E-2001-c1-1-general-file

DDI: `https://osf.io/jtd9h/`

Additional documentation (PDFs): `https://osf.io/hkuy3/`

Value

`cchs2001_p`	a data frame

Source

`http://www23.statcan.gc.ca/imdb/p2SV.pl?Function=getSurvey&Id=3359`

Examples

```
data(cchs2001_p)
str(cchs2001_p)
```

`cchs2003_p` *2003 CCHS PUMF subset data (200 respondents)*

Description

This is a subset of 200 observations from the 2003 cycle of the Canadian Community Health Survey (CCHS) Public Use Microdata file (PUMF) dataset. The CCHS survey is conducted by Statistics Canada.

Details

See here for the open license. Source from Statistics Canada, Canadian Community Health Survey PUMF, accessed Jan 2020. Reproduced and distributed on an "as is" basis with the permission of Statistics Canada.

Long name: cchs-82M0013-E-2003-c2-1-General File

DDI: `https://osf.io/nzq37/`

Additional documentation (PDFs): `https://osf.io/hkuy3/`

Value

`cchs2003_p`	a data frame

Source

`http://www23.statcan.gc.ca/imdb/p2SV.pl?Function=getSurvey&Id=4995`

Examples

```
data(cchs2003_p)
str(cchs2003_p)
```

`cchs2005_p` *2005 CCHS PUMF subset data (200 respondents)*

Description

This is a subset of 200 observations from the 2005 cycle of the Canadian Community Health Survey (CCHS) Public Use Microdata file (PUMF) dataset. The CCHS survey is conducted by Statistics Canada.

Details

See here for the open license. Source from Statistics Canada, Canadian Community Health Survey PUMF, accessed Jan 2020. Reproduced and distributed on an "as is" basis with the permission of Statistics Canada.

Long name: cchs-82M0013-E-2005-c3-1-main-file

DDI: `https://osf.io/35mhq/`

Additional documentation (PDFs): `https://osf.io/hkuy3/`

Value

`cchs2005_p`	a data frame

Source

`http://www23.statcan.gc.ca/imdb/p2SV.pl?Function=getSurvey&Id=22642`

Examples

```
data(cchs2005_p)
str(cchs2005_p)
```

`cchs2007_2008_p` *2007-2008 CCHS PUMF subset data (200 respondents)*

Description

This is a subset of 200 observations from the 2007-2008 cycle of the Canadian Community Health Survey (CCHS) Public Use Microdata file (PUMF) dataset. The CCHS survey is conducted by Statistics Canada.

Details

See here for the open license. Source from Statistics Canada, Canadian Community Health Survey PUMF, accessed Jan 2020. Reproduced and distributed on an "as is" basis with the permission of Statistics Canada.

Long name: cchs-E-2007-2008-AnnualComponent

DDI: `https://osf.io/emzsp/`

Additional documentation (PDFs): `https://osf.io/hkuy3/`

Value

`cchs2007_2008_p`	a data frame

Source

`http://www23.statcan.gc.ca/imdb/p2SV.pl?Function=getSurvey&Id=29539`

Examples

```
data(cchs2007_2008_p)
str(cchs2007_2008_p)
```

`cchs2009_2010_p` *2009-2010 CCHS PUMF subset data (200 respondents)*

Description

This is a subset of 200 observations from the 2009-2010 cycle of the Canadian Community Health Survey (CCHS) Public Use Microdata file (PUMF) dataset. The CCHS survey is conducted by Statistics Canada.

Details

See here for the open license. Source from Statistics Canada, Canadian Community Health Survey PUMF, accessed Jan 2020. Reproduced and distributed on an "as is" basis with the permission of Statistics Canada.

Long name: CCHS-82M0013-E-2009-2010-Annualcomponent

DDI: `https://osf.io/ynzpe/`

Additional documentation (PDFs): `https://osf.io/hkuy3/`

Value

`cchs2009_2010_p`	a data frame

Source

`http://www23.statcan.gc.ca/imdb/p2SV.pl?Function=getSurvey&Id=67251`

Examples

```
data(cchs2009_2010_p)
str(cchs2009_2010_p)
```

`cchs2010_p` *2010 CCHS PUMF subset data (200 respondents)*

Description

This is a subset of 200 observations from the 2010 cycle of the Canadian Community Health Survey (CCHS) Public Use Microdata file (PUMF) dataset. The CCHS survey is conducted by Statistics Canada.

Details

NOTE: this subset of respondents may also be in the 2009-2010 PUMF subset. Please see the "CCHS datasets that overlap each other" article to see how the two datasets contain overlap.

See here for the open license. Source from Statistics Canada, Canadian Community Health Survey PUMF, accessed Jan 2020. Reproduced and distributed on an "as is" basis with the permission of Statistics Canada.

Long name: CCHS-82M0013-E-2010-AnnualComponent

DDI: `https://osf.io/7stpz/`

Additional documentation (PDFs): `https://osf.io/hkuy3/`

Value

`cchs2010_p`	a data frame

Source

`http://www23.statcan.gc.ca/imdb/p2SV.pl?Function=getSurvey&Id=81424`

Examples

```
data(cchs2010_p)
str(cchs2010_p)
```

`cchs2011_2012_p` *2011-2012 CCHS PUMF subset data (200 respondents)*

Description

This is a subset of 200 observations from the 2011-2012 cycle of the Canadian Community Health Survey (CCHS) Public Use Microdata file (PUMF) dataset. The CCHS survey is conducted by Statistics Canada.

Details

See here for the open license. Source from Statistics Canada, Canadian Community Health Survey PUMF, accessed Jan 2020. Reproduced and distributed on an "as is" basis with the permission of Statistics Canada.

Long name: cchs-82M0013-E-2011-2012-Annual-component

DDI: `https://osf.io/zk2vw/`

Additional documentation (PDFs): `https://osf.io/hkuy3/`

Value

`cchs2011_2012_p`	a data frame

Source

`http://www23.statcan.gc.ca/imdb/p2SV.pl?Function=getSurvey&Id=114112`

Examples

```
data(cchs2011_2012_p)
str(cchs2011_2012_p)
```

`cchs2012_p` *2012 CCHS PUMF subset data (200 respondents)*

Description

This is a subset of 200 observations from the 2012 cycle of the Canadian Community Health Survey (CCHS) Public Use Microdata file (PUMF) dataset. The CCHS survey is conducted by Statistics Canada.

Details

NOTE: this subset of respondents may also be in the 2011-2012 PUMF subset. Please see the "CCHS datasets that overlap each other" article to see how the two datasets contain overlap.

See here for the open license. Source from Statistics Canada, Canadian Community Health Survey PUMF, accessed Jan 2020. Reproduced and distributed on an "as is" basis with the permission of Statistics Canada.

Long name: cchs-82M0013-E-2012-Annual-component

DDI: `https://osf.io/sbem8/`

Additional documentation (PDFs): `https://osf.io/hkuy3/`

Value

`cchs2012_p`	a data frame

Source

http://www23.statcan.gc.ca/imdb/p2SV.pl?Function=getSurvey&Id=135927

Examples

```
data(cchs2012_p)
str(cchs2012_p)
```

`cchs2013_2014_p` *2013-2014 CCHS PUMF subset data (200 respondents)*

Description

This is a subset of 200 observations from the 2013-2014 cycle of the Canadian Community Health Survey (CCHS) Public Use Microdata file (PUMF) dataset. The CCHS survey is conducted by Statistics Canada.

Details

See here for the open license. Source from Statistics Canada, Canadian Community Health Survey PUMF, accessed Jan 2020. Reproduced and distributed on an "as is" basis with the permission of Statistics Canada.

Long name: cchs-82M0013-E-2013-2014-Annual-component

DDI: https://osf.io/gy25d/

Additional documentation (PDFs): https://osf.io/hkuy3/

Value

`cchs2013_2014_p`	a data frame

Source

http://www23.statcan.gc.ca/imdb/p2SV.pl?Function=getSurvey&Id=144170

Examples

```
data(cchs2013_2014_p)
str(cchs2013_2014_p)
```

`cchs2014_p` *2014 CCHS PUMF subset data (200 respondents)*

Description

This is a subset of 200 observations from the 2014 cycle of the Canadian Community Health Survey (CCHS) Public Use Microdata file (PUMF) dataset. The CCHS survey is conducted by Statistics Canada.

Details

NOTE: this subset of respondents may also be in the 2013-2014 PUMF subset. Please see the "CCHS datasets that overlap each other" article to see how the two datasets contain overlap.

See here for the open license. Source from Statistics Canada, Canadian Community Health Survey PUMF, accessed Jan 2020. Reproduced and distributed on an "as is" basis with the permission of Statistics Canada.

Long name: cchs-82M0013-E-2014-Annual-component

DDI: `https://osf.io/tbmdn/`

Additional documentation (PDFs): `https://osf.io/hkuy3/`

Value

`cchs2014_p`	a data frame

Source

`http://www23.statcan.gc.ca/imdb/p2SV.pl?Function=getSurvey&Id=164081`

Examples

```
data(cchs2014_p)
str(cchs2014_p)
```

`compare_value_based_on_interval` *Compare Value Based On Interval*

Description

Compare values on the scientific notation interval

Usage

```
compare_value_based_on_interval(
  left_boundary,
  right_boundary,
  data,
  compare_columns,
  interval
)
```

Arguments

`left_boundary`	the min value
`right_boundary`	the max value
`data`	the data that contains values being compared
`compare_columns`	The columns inside data being checked
`interval`	The scientific notation interval

Value

a boolean vector containing true for rows where the comparison is true

`food_insecurity_der` *Food insecurity*

Description

NOTE: this is not a function.

This is a derived variable that uses the different food insecurity variables from all CCHS cycles to generate food_insecurity_der that is harmonized across all cycles. food_insecurity_der is a categorical variable with two categories:

1. no food insecurity in the last 12 months
2. food insecurity in the last 12 months

Usage

```
food_insecurity_der(FINF1, FSCDHFS, FSCDHFS2)
```

Arguments

`FINF1`	variable used in 2001 and 2003 survey cycles indicating food insecurity in the past 12 months
`FSCDHFS`	variable used in the 2005 survey cycle measuring food insecurity & hunger in the last 12 months
`FSCDHFS2`	variable used in 2007-2014 survey cycles measuring household food insecurity in the last 12 months

Details

Food insecurity is measured differently across CCHS cycles. In 2001 and 2003, FINF1 is used; in 2005, FSCDHFS is used; and in 2007 to 2014, FSCDHFS2 is used. Each variable examines food insecurity in the household over the past 12 months, but use different base variables to derive food insecurity.

If you are using cchsflow for CCHS survey years that use consistent food insecurity variables, it is appropriate to use FINF1, FSCDHFS, or FSCDHFS2 that are available on cchsflow. If you are using cchsflow for only the 2001 and 2003 cycles, it is appropriate to use FINF1. If you are using cchsflow for only the 2005 cycle, FSCDHFS is appropriate. If you are using cchsflow for cycles between 2007 and 2014, FSCDHFS2 is appropriate. For multiple CCHS survey years that do not use the same food insecurity variables (i.e. using cchsflow for years 2001 to 2007), food_insecurity_der is recommended.

Examples

```
library(cchsflow)
?food_insecurity_der
```

GEN_02A2 *Satisfaction with life (GEN_02A/GEN_02A2)*

Description

NOTE: this is not a function.

These are two variables asked in the CCHS that asks respondents to rate their satisfaction with their lives. The variable GEN_02A is a categorical variable with 5 categories:

1. Very satisfied
2. Satisfied
3. Neither satisfied nor unsatisfied
4. Dissatisfied
5. Very dissatisfied

The GEN_02A2 is a continuous variable from 0 to 10, where 0 represents very dissatisfied and 10 represents very satisfied.

Usage

```
GEN_02A2(GEN_02A, GEN_02A2)
```

Arguments

`GEN_02A`	- categorical life satisfaction variable asked from 2003-2007
`GEN_02A2`	- continuous life satisfaction variable asked from 2009-2014, and derived for 2003-2007

Details

GEN_02A was asked to respondents in the 2003, 2005, and 2007-2008 CCHS survey cycles; while GEN_02A2 was asked to respondents in CCHS survey cycles from 2009 to 2014. To harmonize GEN_02A2 across more cycles, GEN_02A2 was derived for earlier cycles by converting GEN_02A values to match the scale used in GEN_02A2. The very satisfied category was converted to a score of 10; the satisfied category was converted to a score of 7; the neither satisfied nor unsatisfied category was converted to a score of 5; the dissatisfied category was converted to a score of 2; and the very dissatisfied category was converted to a score of 0.

When using earlier CCHS cycles (2003-2007), it is appropriate to use GEN_02A. When using multiple CCHS cycles that include cycles from 2009-2014, GEN_02A2 is recommended.

Examples

```
library(cchsflow)
?GEN_02A2
```

---

get_data_variable_name *Get Data Variable Name*

Description

Retrieves the name of the column inside data to use for calculations

Usage

```
get_data_variable_name(
  data_name,
  data,
  row_being_checked,
  variable_being_checked
)
```

Arguments

`data_name`	name of the database being checked
`data`	database being checked
`row_being_checked`	the row from variable details that contains information on this variables
`variable_being_checked`	the name of the recoded variable

Value

the data equivalent of variable_being_checked

if_else2 *if_else2*

Description

Custom ifelse function that evaluates missing (NA) values. If the logical argument (x) compares to a value that is ‘NA‘, it is set to ‘FALSE‘

Usage

```
if_else2(x, a, b)
```

Arguments

x	A logical argument
a	value if ‘x‘ is ‘TRUE‘
b	value if ‘x‘ is ‘FALSE‘

Details

unlike the base ifelse() function, if_else2() is able to evaluate NA as either a or b. In base ifelse(), anything compared to NA will produce NA, which can break a function. When dealing with large datasets like the CCHS, there are many missing (NA) values. That means a special ifelse function like if_else2() is needed in order for other functions to not break

Value

a or b based on the evaluation of x

Examples

```
age <- 12
status <- if_else2((age < 18), "child", "invalid age")
print(status)

age <- NA
status <- if_else2((age < 18), "child", "invalid age")
print(status)
```

`is_equal` *is equal*

Description

Function to compare even with NA present This function returns TRUE wherever elements are the same, including NA's, and false everywhere else.

Usage

```
is_equal(v1, v2)
```

Arguments

v1	variable 1
v2	variable 2

Value

boolean value of whether or not v1 and v2 are equal

Examples

```
library(cchsflow)
is_equal(1,2)
# FALSE

is_equal(1,1)
# TRUE

1==NA
# NA

is_equal(1,NA)
# FALSE

NA==NA
# NA

is_equal(NA,NA)
# TRUE
```

label_data *label_data*

Description

Attaches labels to the DataToLabel to preserve metadata

Usage

```
label_data(label_list, data_to_label)
```

Arguments

`label_list`	the label list object that contains extracted labels from variable details
`data_to_label`	The data that is to be labeled

Value

Returns labeled data

multiple_conditions_fun1 *Multiple chronic conditions (5 chronic conditions)*

Description

This function generates a derived variable (multiple_conditions) that counts the number of chronic conditions a respondent has. This function takes 5 CCHS-defined conditions (heart disease, cancer, stroke, bowel disorder, and arthritis), and well one derived variable (respiratory condition) to count the number of conditions a respondent has.

Usage

```
multiple_conditions_fun1(
  CCC_121,
  CCC_131,
  CCC_151,
  CCC_171,
  resp_condition_der,
  CCC_051
)
```

Arguments

`CCC_121`	variable indicating if respondent has heart disease (1 = respondent has heart disease, 2 = respondent does not have heart disease)
`CCC_131`	variable indicating if respondent has active cancer (1 = respondent has active cancer, 2 = respondent does not have active cancer)
`CCC_151`	variable indicating if respondent suffers from the effects of a stroke (1 = respondent suffers from stroke effects, 2 = respondent does not suffer from stroke effects)
`CCC_171`	variable indicating if respondent has a bowel disorder (1 = respondent has bowel disorder, 2 = respondent does not have a bowel disorder)
`resp_condition_der`	derived variable indicating if respondent has a respiratory condition (1 = respondent is over the age of 35 and has a respiratory condition, 2 = respondent is under the age of 35 and has a respiratory conditions, 3 = respondent does not have a respiratory condition). See `resp_condition_fun1` for documentation on how variable was derived.
`CCC_051`	variable indicating if respondent has arthritis or rheumatism (1 = respondent has arthritis or rheumatism, 2 = respondent does not have arthritis or rheumatism)

Details

mood disorder (CCC_280) was not asked to respondents in the 2001 CCHS survey cycle. This mean respondents in this cycle will only be able to have a maximum of 6 chronic conditions as opposed to 7 for respondents in other cycles. `multiple_conditions_fun2` is used for CCHS cycles from 2003 to 2014.

Value

A categorical variable indicating the number of chronic conditions a respondent has. Respondents with 5 or more conditions are grouped in the "5+" category.

See Also

`multiple_conditions_fun2`

Examples

```
# Using rec_with_table() to generate multiple_conditions in a CCHS
# cycle.

# multiple_conditions_fun1() is specified in variable_details.csv along with
# the CCHS variables and cycles included.

# To generate multiple_conditions, use rec_with_table() and specify the
# multiple_conditions, along with the variables that are derived from it.
# Since resp_condition_der is also a derived variable, you will have to
# specify the variables that are derived from it. In this example, data
# from the 2001 CCHS will be used, so DHHGAGE_cont, CCC_091, and CCC_91A,
 # and CCC_031 will be specified along with resp_condition_der.

library(cchsflow)
 conditions_2001 <- suppressWarnings(rec_with_table(cchs2001_p,
 c("DHHGAGE_cont", "CCC_091",
 "CCC_91A", "CCC_031", "CCC_121","CCC_131","CCC_151", "CCC_171","CCC_280",
 "resp_condition_der","CCC_051", "multiple_conditions")))

 head(conditions_2001)

 # Generating multiple_conditions with user inputted values
 # Let's say you are an individual that has heart disease, bowel disorder,
 # and arthritis. multiple_conditions_fun1() can be used to count the number
 # of chronic conditions you have

library(cchsflow)
 num_conditions <- multiple_conditions_fun1(CCC_121 = 1, CCC_131 = 2,
 CCC_151 = 2, CCC_171 = 1, resp_condition_der = 3, CCC_051 = 1)

print(num_conditions)
```

---

multiple_conditions_fun2

Multiple chronic conditions (6 chronic conditions)

Description

This function generates a derived variable (multiple_conditions) that counts the number of chronic conditions a respondent has. This function takes 6 CCHS-defined conditions (heart disease, cancer, stroke, bowel disorder, mood disorder and arthritis), and well one derived variable (respiratory condition) to count the number of conditions a respondent has.

Usage

```
multiple_conditions_fun2(
  CCC_121,
  CCC_131,
  CCC_151,
  CCC_171,
  CCC_280,
  resp_condition_der,
  CCC_051
)
```

Arguments

CCC_121	variable indicating if respondent has heart disease (1 = respondent has heart disease, 2 = respondent does not have heart disease)

`CCC_131`	variable indicating if respondent has active cancer (1 = respondent has active cancer, 2 = respondent does not have active cancer)
`CCC_151`	variable indicating if respondent suffers from the effects of a stroke (1 = respondent suffers from stroke effects, 2 = respondent does not suffer from stroke effects)
`CCC_171`	variable indicating if respondent has a bowel disorder (1 = respondent has bowel disorder, 2 = respondent does not have a bowel disorder)
`CCC_280`	variable indicating if respondent has a mood disorder (1 = respondent has a mood disorder, 2 = respondent does not have a mood disorder. Note, variable was not asked to respondents in the 2001 CCHS survey cycle.
`resp_condition_der`	derived variable indicating if respondent has a respiratory condition. (1 = respondent is over the age of 35 and has a respiratory condition, 2 = respondent is under the age of 35 and has a respiratory conditions, 3 = respondent does not have a respiratory condition). See `resp_condition_fun1` for documentation on how variable was derived.
`CCC_051`	variable indicating if respondent has arthritis or rheumatism (1 = respondent has arthritis or rheumatism, 2 = respondent does not have arthritis or rheumatism)

Details

mood disorder (CCC_280) was not asked to respondents in the 2001 CCHS survey cycle. This mean respondents in this cycle will only be able to have a maximum of 6 chronic conditions as opposed to 7 for respondents in other cycles. `multiple_conditions_fun1` is used for CCHS cycles from 2003 to 2014.

Value

A categorical variable indicating the number of chronic conditions a respondent has. Respondents with 5 or more conditions are grouped in the "5+" category.

See Also

`multiple_conditions_fun1`

Examples

```
# Using rec_with_table() to generate multiple_conditions in a CCHS
# cycle.

# multiple_conditions_fun2() is specified in variable_details.csv along with
# the CCHS variables and cycles included.

# To generate multiple_conditions, use rec_with_table() and specify the
# multiple_conditions, along with the variables that are derived from it.
# Since resp_condition_der is also a derived variable, you will have to
# specify the variables that are derived from it. In this example, data
# from the 2010 CCHS will be used, so DHHGAGE_cont, CCC_091, and CCC_031
# will be specified along with resp_condition_der.
```

```
library(cchsflow)
 conditions_2009_2010 <- suppressWarnings(rec_with_table(cchs2009_2010_p,
 c("DHHGAGE_cont", "CCC_091",
 "CCC_031", "CCC_121","CCC_131","CCC_151", "CCC_171","CCC_280",
 "resp_condition_der","CCC_051", "multiple_conditions")))

 head(conditions_2009_2010)

 # Generating multiple_conditions with user inputted values
 # Let's say you are an individual that has heart disease, bowel disorder,
 # and arthritis. multiple_conditions_fun2() can be used to count the number
 # of chronic conditions you have

library(cchsflow)
 num_conditions <- multiple_conditions_fun2(CCC_121 = 1, CCC_131 = 2,
 CCC_151 = 2, CCC_171 = 1, CCC_280 = 2, resp_condition_der = 3, CCC_051 = 1)

print(num_conditions)
```

pack_years_fun *Smoking pack-years*

Description

This function creates a derived variable (pack_years_der) that measures an individual's smoking pack-years based on various CCHS smoking variables. This is a popular variable used by researchers to quantify lifetime exposure to cigarette use.

Usage

```
pack_years_fun(
  SMKDSTY,
  DHHGAGE_cont,
  SMK_09A_B,
  SMKG09C,
  SMKG203_cont,
  SMKG207_cont,
  SMK_204,
  SMK_05B,
  SMK_208,
  SMK_05C,
  SMKG01C_cont,
  SMK_01A
)
```

Arguments

SMKDSTY	derived variable that classifies an individual's smoking status.
DHHGAGE_cont	continuous age variable.
SMK_09A_B	number of years since quitting smoking. Variable asked to former daily smokers who quit <3 years ago.
SMKG09C	number of years since quitting smoking. Variable asked to former daily smokers who quit >=3 years ago.
SMKG203_cont	age started smoking daily. Variable asked to daily smokers.
SMKG207_cont	age started smoking daily. Variable asked to former daily smokers.
SMK_204	number of cigarettes smoked per day. Variable asked to daily smokers.
SMK_05B	number of cigarettes smoked per day. Variable asked to occasional smokers
SMK_208	number of cigarettes smoked per day. Variable asked to former daily smokers
SMK_05C	number of days smoked at least one cigarette
SMKG01C_cont	age smoked first cigarette
SMK_01A	smoked 100 cigarettes in lifetime (y/n)

Details

pack-years is calculated by multiplying the number of cigarette packs per day (20 cigarettes per pack) by the number of years. Example 1: a respondent who is a current smoker who smokes 1 package of cigarettes for the last 10 years has smoked 10 pack-years. Pack-years is also calculated for former smokers. Example 2: a respondent who started smoking at age 20 years and smoked half a pack of cigarettes until age 40 years smoked for 10 pack-years.

Value

value for smoking pack-years in the Pack_years_der variable

Examples

```
# Using pack_years_fun() to create pack-years values across CCHS cycles
# pack_years_fun() is specified in variable_details.csv along with the CCHS
# variables and cycles included.

# To transform pack_years_der across cycles, use rec_with_table() for each
# CCHS cycle and specify pack_years_der, along with each smoking variable.
# Then by using bind_rows(), you can combine pack_years_der across cycles

library(cchsflow)

pack_years2009_2010 <- rec_with_table(
  cchs2009_2010_p, c(
    "SMKDSTY", "DHHGAGE_cont", "SMK_09A_B", "SMKG09C",
    "SMKG203_cont", "SMKG207_cont", "SMK_204", "SMK_05B", "SMK_208",
    "SMK_05C", "SMK_01A", "SMKG01C_cont", "pack_years_der"
  )
)
```

```
head(pack_years2009_2010)

pack_years2011_2012 <- rec_with_table(
  cchs2011_2012_p,c(
    "SMKDSTY", "DHHGAGE_cont", "SMK_09A_B", "SMKG09C",
    "SMKG203_cont", "SMKG207_cont", "SMK_204", "SMK_05B", "SMK_208",
    "SMK_05C", "SMK_01A", "SMKG01C_cont", "pack_years_der"
  )
)

tail(pack_years2011_2012)

combined_pack_years <- suppressWarnings(bind_rows(pack_years2009_2010,
 pack_years2011_2012))

head(combined_pack_years)
tail(combined_pack_years)
```

pct_time_fun *Percent time in Canada*

Description

This function creates a derived variable (pct_time_der) that provides an estimated percentage of the time a person's life was spent in Canada.

Usage

```
pct_time_fun(DHHGAGE_cont, SDCGCBG, SDCGRES)
```

Arguments

`DHHGAGE_cont`	continuous age variable.
`SDCGCBG`	whether or not someone was born in Canada (1 - born in Canada, 2 - born outside Canada)
`SDCGRES`	how long someone has lived in Canada. Note: in the PUMF CCHS datasets, this is a categorical variable with two categories (1 - 0-9 years; 2 - 10+ years).

Value

Numeric value that is a fraction between 0 and 1 that represents percentage of a respondent's time in Canada

Note

Since SDCGRES is a categorical variable measuring length of time, we've set midpoints in the function. A respondent identified as being in Canada for 0-9 years is assigned a value of 4.5 years, and someone who has been in Canada for over 10 years is assigned a value of 15 years.

Examples

```
# Using pct_time_fun() to create percent time values between CCHS cycles
# pct_time_fun() is specified in variable_details.csv along with the CCHS
# variables and cycles included.

# To transform pct_time_der across cycles, use rec_with_table() for each CCHS
# cycle and specify pct_time_der, along with age (DHHGAGE_cont), whether or
# not someone was born in Canada (SDCGCBG), how long someone has lived in
# Canada (SDCGRES). Then by using bind_rows(), you can combine Pct_time_der
# across cycles

library(cchsflow)
pct_time2009_2010 <- rec_with_table(
  cchs2009_2010_p, c(
    "DHHGAGE_cont", "SDCGCBG",
    "SDCGRES", "pct_time_der"
  )
)
head(pct_time2009_2010)

pct_time2011_2012 <- rec_with_table(
  cchs2011_2012_p,  c(
    "DHHGAGE_cont", "SDCGCBG",
    "SDCGRES", "pct_time_der"
  )
)
tail(pct_time2011_2012)

combined_pct_time <- bind_rows(pct_time2009_2010, pct_time2011_2012)
head(combined_pct_time)
tail(combined_pct_time)

# Using pct_time_fun() to generate a value for percent time spent in Canada
# with user inputted values Let's say you are 27 years old who was born
# outside of Canada and have been living in Canada for less than 10 years.
# Your estimated percent time spent in Canada can be calculated as follows:

pct_time <- pct_time_fun(DHHGAGE_cont = 27, SDCGCBG = 2, SDCGRES = 1)

print(pct_time)
```

RACDPAL_fun *Participation and Activity Limitation*

Description

This is a derived variable used in the CCHS (RACDPAL) to classify respondents according to the frequency with which they experience activity limitations due to disability.

Usage

```
RACDPAL_fun(RAC_1, RAC_2A, RAC_2B, RAC_2C)
```

Arguments

RAC_1	Has difficulty with activities due to disability
RAC_2A	Reduction in activities at home due to disability
RAC_2B	Reduction in activities at school or work due to disability
RAC_2C	Reduction in other activities

Details

This derived variable is generated in CCHS cycles 2003-2014. The 2001 CCHS cycle, however, contains the same base variables used to derive this variable. To include respondents in the 2001 CCHS cycle, this custom function was created using the same derivation conditions used in later cycles.

Value

the CCHS derived variable RACDPAL with 3 categories:

1. Sometimes
2. Often
3. Never

Examples

```
# Using RACDPAL_fun() to transform RACDPAL in 2001.
# RACDPAL_fun() is specified in variable_details.csv along with the
# CCHS variables and cycles included.

# To transform RACDPAL, use rec_with_table() for each the 2001 cycle
# and specify RACDPAL, along with the various ADL variables.

library(cchsflow)

RACDPAL_2001 <- rec_with_table(
  cchs2001_p, c(
    "RAC_1", "RAC_2A", "RAC_2B", "RAC_2C", "RACDPAL"
  )
)

head(RACDPAL_2001)

# Note: In other CCHS cycles you only need to specify RACDPAL as the variable
# was included in those survey cycles.

# Using RACDPAL_fun() with user inputted data.

# Let's say you're an individual that sometimes has difficulties with
# activities due to disability, sometimes has a reduction in activities at
# home, often has a reduction at school or work, and never has a reduction
# in other activities. Your participation and activity limitation can be
# determined as follows:

library(cchsflow)
RACDPAL <- RACDPAL_fun(1, 1, 2, 3)
print(RACDPAL)
```

recode_columns *recode_columns*

Description

Recodes columns from passed row and returns just table with those columns and same rows as the data

Usage

```
recode_columns(
  data,
  variables_to_process,
  data_name,
  log,
  print_note,
  else_default
)
```

Arguments

`data`	The source database
`variables_to_process`	rows from variable details that are applicable to this DB
`data_name`	Name of the database being passed
`log`	The option of printing log
`print_note`	the option of printing the note columns
`else_default`	default else value to use if no else is present

Value

Returns recoded and labeled data

rec_with_table *Recode with Table*

Description

Recode with Table is responsible for recoding values of a dataset based on the specifications in variable_details.

Usage

```
rec_with_table(
  data,
  variables = NULL,
  database_name = NULL,
  variable_details = NULL,
  else_value = NA,
  append_to_data = FALSE,
  log = FALSE,
  notes = TRUE,
  var_labels = NULL,
  custom_function_path = NULL,
  attach_data_name = FALSE
)
```

Arguments

`data`	A dataframe containing the variables to be recoded.
`variables`	character vector containing variable names to recode or a variables csv containing additional variable info
`database_name`	String, the name of the dataset containing the to be recoded.
`variable_details`	A dataframe containing the specifications (rules) for recoding.
`else_value`	Value (string, number, integer, logical or NA) that is used to replace any values that are outside the specified ranges (no rules for recoding).
`append_to_data`	Logical, if `TRUE` (default), recoded variables will be appended to the data.
`log`	Logical, if `FALSE` (default), a log of recoding will not be printed.
`notes`	Logical, if `FALSE` (default), will not print the content inside the ‘Note“ column of the variable being recoded.
`var_labels`	labels vector to attach to variables in variables
`custom_function_path`	path to location of the function to load
`attach_data_name`	to attach name of database to end table

Details

The variable_details dataframe needs the following variables to function:

variable name of new (mutated) variable that is recoded

toType type the variable is being recoded to *cat = categorical, cont = continues*

databaseStart name of dataframe with original variables to be recoded

variableStart name of variable to be recoded

fromType variable type of start variable. *cat = categorical or factor variable cont = continuous variable (real number or integer)*

recTo Value to recode to

recFrom Value/range being recoded from

Each row in *variable_details* comprises one category in a newly transformed variable. The rules for each category the new variable are a string in *recFrom* and value in *recTo*. These recode pairs are the same syntax as *sjmisc::rec()*, except in *sjmisc::rec()* the pairs are a string for the function attribute *rec* =, separated by '='. For example in *rec_w_table variable_details$recFrom = 2; variable_details$recTo = 4* is the same as *sjmisc::rec(rec = "2=4").* the pairs are obtained from the RecFrom and RecTo columns

recode pairs each recode pair is row. see above example or *PBC-variableDetails.csv*

multiple values multiple old values that should be recoded into a new single value may be separated with comma, e.g. *recFrom = "1,2"; recTo = 1*

value range a value range is indicated by a colon, e.g. *recFrom= "1:4"; recTo = 1* (recodes all values from 1 to 4 into 1)

value range for doubles for double vectors (with fractional part), all values within the specified range are recoded; e.g. *recFrom = "1:2.5'; recTo = 1* recodes 1 to 2.5 into 1, but 2.55 would not be recoded (since it's not included in the specified range)

"min"* and *"max" minimum and maximum values are indicates by *min* (or *lo*) and *max* (or *hi*), e.g. *recFrom = "min:4"; recTo = 1* (recodes all values from minimum values of *x* to 4 into 1)

"else" all other values, which have not been specified yet, are indicated by *else*, e.g. *recFrom = "else"; recTo = NA* (recode all other values (not specified in other rows) to "NA")

"copy" the *"else"*-token can be combined with *copy*, indicating that all remaining, not yet recoded values should stay the same (are copied from the original value), e.g. *recFrom = "else"; recTo = "copy"*

***NA*'s** *NA* values are allowed both as old and new value, e.g. *recFrom "NA"; recTo = 1. or "recFrom = "3:5"; recTo = "NA"* (recodes all NA into 1, and all values from 3 to 5 into NA in the new variable)

Value

a dataframe that is recoded according to rules in variable_details.

Examples

```
library(cchsflow)
bmi2001 <- rec_with_table(
  data = cchs2001_p, c(
    "HWTGHTM",
    "HWTGWTK", "HWTGBMI_der"
  )
)

head(bmi2001)

bmi2011_2012 <- rec_with_table(
  data = cchs2011_2012_p,  c(
    "HWTGHTM",
    "HWTGWTK", "HWTGBMI_der"
  )
)

tail(bmi2011_2012)

combined_bmi <- bind_rows(bmi2001, bmi2011_2012)
head(combined_bmi)
tail(combined_bmi)
```

`resp_condition_fun1` *resp_condition_fun1*

Description

This is one of 3 functions used to create a derived variable (resp_condition_der) that determines if a respondents has a respiratory condition. 3 different functions have been created to account for the fact that different respiratory variables are used across CCHS cycles. This function is for CCHS cycles (2009-2014) that only use COPD and Emphysema as a combined variable. Asthma is used across CCHS cycles as a separate variable.

Usage

```
resp_condition_fun1(DHHGAGE_cont, CCC_091, CCC_031)
```

Arguments

`DHHGAGE_cont`	continuous age variable.
`CCC_091`	variable indicating if respondent has either COPD or Emphysema
`CCC_031`	variable indicating if respondent has asthma

Value

a categorical variable (resp_condition_der) with 3 levels:

1. respondent is over the age of 35 and has a respiratory condition
2. respondent is under the age of 35 and has a respiratory condition
3. respondent does not have a respiratory condition

See Also

resp_condition_fun2, resp_condition_fun3

Examples

```
# Using resp_condition_fun1() to create values across CCHS cycles
# (2009-2014) resp_condition_fun1() is specified in
# variable_details.csv along with the CCHS variables and cycles included.

# To transform resp_condition_der, use rec_with_table() for each CCHS cycle
# and specify resp_condition_der, along with the various respiratory
# variables. Then by using bind_rows() you can combine resp_condition_der
# across cycles.

library(cchsflow)

resp2009_2010 <- suppressWarnings(rec_with_table(
  cchs2009_2010_p,  c(
    "DHHGAGE_cont", "CCC_091", "CCC_031",
    "resp_condition_der"
  )
))

head(resp2009_2010)

resp2011_2012 <- suppressWarnings(rec_with_table(
  cchs2011_2012_p, c(
    "DHHGAGE_cont", "CCC_091", "CCC_031",
    "resp_condition_der"
  )
))

tail(resp2011_2012)

combined_resp <- suppressWarnings(bind_rows(resp2009_2010, resp2011_2012))

head(combined_resp)
tail(combined_resp)
```

resp_condition_fun2 *resp_condition_fun2*

Description

This is one of 3 functions used to create a derived variable (resp_condition_der) that determines if a respondents has a respiratory condition. This function is for CCHS cycles (2005-2007) that use COPD & Emphysema as separate variables, as well as Bronchitis. Asthma is used across CCHS cycles as a separate variable.

Usage

```
resp_condition_fun2(DHHGAGE_cont, CCC_91E, CCC_91F, CCC_91A, CCC_031)
```

Arguments

DHHGAGE_cont	continuous age variable.
CCC_91E	variable indicating if respondent has emphysema
CCC_91F	variable indicating if respondent has COPD
CCC_91A	variable indicating if respondent has chronic bronchitis
CCC_031	variable indicating if respondent has asthma

Value

a categorical variable (resp_condition_der) with 3 levels:

1. respondent is over the age of 35 and has a respiratory condition
2. respondent is under the age of 35 and has a respiratory condition
3. respondent does not have a respiratory condition

See Also

resp_condition_fun1, resp_condition_fun3

Examples

```
# Using resp_condition_fun2() to create values across CCHS cycles
# (2005-2007) resp_condition_fun2() is specified in
# variable_details.csv along with the CCHS variables and cycles included.

# To transform resp_condition_der, use rec_with_table() for each CCHS cycle
# and specify resp_condition_der, along with the various respiratory
# variables. Then by using bind_rows() you can combine resp_condition_der
# across cycles.

library(cchsflow)
```

```
resp2005 <- suppressWarnings(rec_with_table(
  cchs2005_p, c(
    "DHHGAGE_cont", "CCC_91E", "CCC_91F", "CCC_91A", "CCC_031",
    "resp_condition_der"
  )
))

head(resp2005)

resp2007_2008 <- suppressWarnings(rec_with_table(
  cchs2007_2008_p,  c(
    "DHHGAGE_cont", "CCC_91E", "CCC_91F", "CCC_91A", "CCC_031",
    "resp_condition_der"
  )
))

tail(resp2007_2008)

combined_resp <- suppressWarnings(bind_rows(resp2005, resp2007_2008))

head(combined_resp)
tail(combined_resp)
```

resp_condition_fun3 *resp_condition_fun3*

Description

This is one of 3 functions used to create a derived variable (resp_condition_der) that determines if a respondents has a respiratory condition. This function for CCHS cycles (2001-2003) that use COPD and Emphysema as a combined variable, as well as Bronchitis. Asthma is used across CCHS cycles as a separate variable.

Usage

```
resp_condition_fun3(DHHGAGE_cont, CCC_091, CCC_91A, CCC_031)
```

Arguments

DHHGAGE_cont	continuous age variable.
CCC_091	variable indicating if respondent has either COPD or Emphysema
CCC_91A	variable indicating if respondent has chronic bronchitis
CCC_031	variable indicating if respondent has asthma

Value

a categorical variable (resp_condition_der) with 3 levels:

1. respondent is over the age of 35 and has a respiratory condition
2. respondent is under the age of 35 and has a respiratory condition
3. respondent does not have a respiratory condition

See Also

resp_condition_fun1, resp_condition_fun2

Examples

```
# Using resp_condition_fun3() to create values across CCHS cycles
# (2001-2003) resp_condition_fun3() is specified in
# variable_details.csv along with the CCHS variables and cycles included.

# To transform resp_condition_der, use rec_with_table() for each CCHS cycle
# and specify resp_condition_der, along with the various respiratory
# variables. Then by using bind_rows() you can combine resp_condition_der
# across cycles.

library(cchsflow)

resp2001 <- suppressWarnings(rec_with_table(
  cchs2001_p, c(
    "DHHGAGE_cont", "CCC_091", "CCC_91A", "CCC_031",
    "resp_condition_der"
  )
))

head(resp2001)

resp2003 <- suppressWarnings(rec_with_table(
  cchs2003_p,c(
    "DHHGAGE_cont", "CCC_091", "CCC_91A", "CCC_031",
    "resp_condition_der"
  )
))

tail(resp2003)

combined_resp <- suppressWarnings(bind_rows(resp2001, resp2003))

head(combined_resp)
tail(combined_resp)
```

`set_data_labels` *Set Data Labels*

Description

sets labels for passed database, Uses the names of final variables in variable_details/variables_sheet as well as the labels contained in the passed dataframes

Usage

```
set_data_labels(data_to_label, variable_details, variables_sheet = NULL)
```

Arguments

`data_to_label`	newly transformed dataset
`variable_details`	variable_details.csv
`variables_sheet`	variables.csv

Value

labeled data_to_label

Examples

```
library(cchsflow)
library(sjlabelled)
bmi2001 <- rec_with_table(
 cchs2001_p, c(
    "HWTGHTM",
    "HWTGWTK", "HWTGBMI_der"
  )
)

bmi2003 <- rec_with_table(
  cchs2003_p, c(
    "HWTGHTM",
    "HWTGWTK", "HWTGBMI_der"
  )
)

combined_bmi <- bind_rows(bmi2001, bmi2003)

get_label(combined_bmi)

labeled_combined_data <- set_data_labels(combined_bmi,
 variable_details,
 variables)
```

```
get_label(labeled_combined_data)
```

`variables` *variables.csv*

Description

This dataset lists all the variables that are present in cchsflow.

Details

See the below link for more details about how the worksheet is structured https://big-life-lab.github.io/cchsflow/articles/variables_sheet.html

Value

`variables`	a data frame

Examples

```
data(variables)
str(variables)
```

`variable_details` *variable_details.csv*

Description

This dataset provides details on how variables are recoded in cchsflow.

Details

See the below link for more details about how the worksheet is structured https://big-life-lab.github.io/cchsflow/articles/variable_details.html

Value

`variable_details`	a data frame

Examples

```
data(variable_details)
str(variable_details)
```

Index