Package 'NVIdb'

May 3, 2024

Title Tools to facilitate the use of NVI's databases

Version 0.11.3

Date 2024-05-03

Description Provides tools to facilitate downloading and processing of data from the Norwegian Veterinary Institute's databases, in particular PJS and EOS. The package comprises several categories of functions:
1) Manage credentials (i.e. password and username),
2) Login functions for database services,
3) Select PJS-data,
4) Initial cleaning of PJS-data,
5) Read, copy and update various in-house data registers,
6) Translate codes into descriptions.
NVIdb is dependant of NVIconfig which has to be installed manually from GitHub.

URL https://github.com/NorwegianVeterinaryInstitute/NVIdb

BugReports https://github.com/NorwegianVeterinaryInstitute/NVIdb/issues

Depends R (>= 3.5.0)

License BSD_3_clause + file LICENSE

Imports askpass,
checkmate,
data.table,
DBI,
dplyr,
keyring,
magrittr,
odbc,
RODBC,
RPostgreSQL,
snakecase,
stats,
utils,
NVIcheckmate (>= 0.7.0)

Suggests covr,
desc,
devtools,
knitr,
R.rsp,
remotes,

rmarkdown,
spelling,
testthat,
tibble,
NVIpackager,
NVIrpackages

Remotes NorwegianVeterinaryInstitute/NVIcheckmate,
NorwegianVeterinaryInstitute/NVIrpackages,
NorwegianVeterinaryInstitute/NVIpackager

LazyData true

Encoding UTF-8

Language en-GB

Roxygen list(markdown = FALSE)

RoxygenNote 7.3.1

VignetteBuilder knitr,
R.rsp

R topics documented:

add_kommune_fylke . . . 3
add_lokalitet . . . 5
add_MT_omrader . . . 7
add_PJS_code_description . . . 10
add_poststed . . . 14
add_produsent_properties . . . 17
build_query_hensikt . . . 19
build_query_one_disease . . . 20
build_query_outbreak . . . 22
build_sql_modules . . . 23
choose_PJS_levels . . . 24
copy_Pkode_2_text . . . 26
copy_Prodtilskudd . . . 28
cut_slash . . . 29
exclude_from_PJSdata . . . 30
login . . . 31
NVIdb . . . 34
PJS_code_description_colname . . . 35
PJS_levels . . . 35
read_eos_data . . . 36
read_leveransereg . . . 37
read_varekode . . . 38
remove_credentials . . . 39
retrieve_PJSdata . . . 41
select_PJSdata_for_value . . . 42
set_dir_NVI . . . 43
set_disease_parameters . . . 44
standardize_columns . . . 47
standardize_eos_data . . . 49
standardize_PJSdata . . . 51
transform_code_combinations . . . 52

Index 55

`add_kommune_fylke` *Manage translation from komnr to kommune, fylke and current komnr*

Description

Function to add columns with kommune (name), fylkenr, fylke (name), gjeldende_komnr, gjeldende_kommune, gjeldende_fylkenr, and gjeldende_fylke. In addition there are functions to read and copy the translation tables.

Usage

```
add_kommune_fylke(
  data,
  translation_table = kommune_fylke,
  code_column = c("komnr"),
  new_column = c("gjeldende_komnr", "gjeldende_kommune", "gjeldende_fylkenr",
    "gjeldende_fylke"),
  position = "right",
  overwrite = FALSE
)

copy_kommune_fylke(
  filename = list("Kommune_UTF8.csv", "komnr_2_gjeldende_komnr_UTF8.csv",
    "Fylke_UTF8.csv"),
  from_path = paste0(set_dir_NVI("GrunndataLand"), "FormaterteData/"),
  to_path = NULL
)

read_kommune_fylke(
  filename = list("Kommune_UTF8.csv", "komnr_2_gjeldende_komnr_UTF8.csv",
    "Fylke_UTF8.csv"),
  from_path = paste0(set_dir_NVI("GrunndataLand"), "FormaterteData/")
)
```

Arguments

`data`	Data frame with data with a column with old komnr
`translation_table`	Data frame with the translation table for old komnr to current komnr
`code_column`	The name of the column with the old komnr
`new_column`	The name of the new column that should contain the current komnr
`position`	[`character(1)`] The position for the new columns, can be one of c("first", "left", "right", "last", "keep"). The input can be abbreviated, but must be unique, i.e. c("f", "le", "r", "la", "k"). Defaults to "right".
`overwrite`	[`logical(1)`] When the new column(s) already exist, the content in the existing column(s) is replaced by new data if `overwrite = TRUE`. If the new column(s) already exist and `overwrite = FALSE`, an error is issued. Defaults to `FALSE`.

filename	Filename of the translation table for old komnr to current komnr
from_path	Path for the source translation table
to_path	Path for the target translation table when copying the translation table

Details

Data sources, like PJS, may provide data with komnr. These functions manage translating komnr to current komnr, kommune and fylke.

add_kommune_fylke can be used to translate komnr into kommune (name), fylkenr, fylke (name), gjeldende_komnr, gjeldende_kommune, gjeldende_fylkenr, and gjeldende_fylke. The function can also be used to translate fylkenr into fylke (name), gjeldende_fylkenr, and gjeldende_fylke.

One has to ensure that the code in the dataset represents a komnr or fylkenr. The function will translate any 4 and 2 digits that has the same ID as a kommune or fylke, respectively.

Standard name for the kommunenummer is komnr. If the column with the komnr that should be translated has another name, the parameter code_column can be input as a named vector. Standard names for the new columns are c("kommune", "fylkenr", "fylke", "gjeldende_komnr", "gjeldende_kommune", "gjeldende_fylkenr", "gjeldende_fylke"). Likewise, if the new columns should be given other names than, the parameter new_column can be input as a named vector, see examples.

The function uses a premade translation table that is made based on information in PJS adresseregister. The translation table is updated when informed that know there is a need, typically when there have been changes in kommune-structure.

position is used to give the place if the new columns in the data.frame. For position = "right" the new variables are placed to the right of the code_variable. Likewise, for position = "left" the new variables are placed to the left of the code_variable. If position = "first" or position = "last" the new columns are placed first or last, respectively, in the data.frame. A special case occurs for position = "keep" which only has meaning when the new column has the same name as an existing column and overwrite = TRUE. In these cases, the existing column will be overwritten with new data and have the same position.

read_kommune_fylke read the files "komnr_2_gjeldende_komnr_UTF8.csv", "Kommune_UTF8.csv", and "Fylke_UTF8.csv", into a single data frame that can be used by add_kommune_fylke. Standard setting will read in the file from NVI's internal network. If changing the from_path, the function can be used to read the translation files from other directories. This can be useful if having a stand alone app with no connection the NVI's internal network. In other cases, it should be avoided.

copy_kommune_fylke copy the files "komnr_2_gjeldende_komnr_UTF8.csv", "Kommune_UTF8.csv", and "Fylke_UTF8.csv", respectively, to a given directory.

Value

add_kommune_fylke A data frame where one or more of the columns c("kommune", "fylkenr", "fylke", "gjeldende_komnr", "gjeldende_kommune", "gjeldende_fylkenr", "gjeldende_fylke") have been added in the column(s) to the right of the column with the komnr.

read_kommune_fylke A data frame with the original komnr and the corresponding kommune, fylkenr, fylke, and the current komnr, kommune, fylkenr, fylke. If not changing standard input to the function, the standard file at NVI's internal network is read.

copy_kommune_fylke copies the source translation table for komnr to kommune, for old komnr to current komnr, and for fylkenr to fylke to given directory. If the target file already exists, the source file is copied only when it is newer than the target file.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Reading from standard directory at NVI's network
kommune_fylke <- read_kommune_fylke()

# Copy standard file from standard location to the subdirectory Data below the working directory
copy_kommune_fylke(to_path = "./Data/")

# Reading from the subdirectory Data below the working directory
kommune_fylke <- read_kommune_fylke(from_path = "./Data/")

# Add new column with current komnr and kommune
# The variable gammelt_komnr should be translated and the new variables with gjeldende_komnr" and
# "gjeldende_kommune" is named komnr and kommune, respectively.
newdata <- add_kommune_fylke(olddata,
                             translation_table = kommune_fylke,
                             code_column = c("gammelt_komnr" = "komnr"),
                             new_column = c("komnr" = "gjeldende_komnr",
                                            "kommune" = "gjeldende_kommune"))

## End(Not run)
```

---

add_lokalitet *Manage adding extra information to aquaculture sites*

Description

Function to add a column with current aquaculture zone and/or geo-coordinates. In addition there are function to read the translation table.

Usage

```
add_lokalitet(
  data,
  translation_table,
  code_column,
  new_column,
  position = "right",
  overwrite = FALSE
)

read_sonetilhorighet(
  filename = "sonetilhorighet.txt",
  from_path = paste0(set_dir_NVI("EksterneDatakilder"), "Lokreg/FormaterteData/Soner/")
)
```

Arguments

`data`	[`data.frame`] Data with a column with an aquaculture site number ("LokNr")
`translation_table`	[`data.frame`] Table for translating from loknr to the property in question.
`code_column`	[`character(1)`] The column with the coded value. Valid values are one of c("LokNr"). If the column in data has another name, it can be input as a named vector, see examples.
`new_column`	[`character`] The new columns that should be included into the data frame.
`position`	[`character(1)`] The position for the new columns, can be one of c("first", "left", "right", "last", "keep"). The input can be abbreviated, but must be unique, i.e. c("f", "le", "r", "la", "k"). Defaults to "right".
`overwrite`	[`logical(1)`] When the new column(s) already exist, the content in the existing column(s) is replaced by new data if `overwrite = TRUE`. If the new column(s) already exist and `overwrite = FALSE`, an error is issued. Defaults to `FALSE`.
`filename`	[`list`] The filenames of the source files with the tables for generating the translation table.
`from_path`	[`character(1)`] Path for the source files for the translation table.

Details

`add_lokalitet` can be used to add aquaculture zone and/or geo-coordinates to aquaculture sites. The new columns can be one or more of c("sone", "EastUTM_33N_WGS84", "NorthUTM_33N_WGS84", "Longitude_WGS84", "Latitude_WGS84"). If the new columns in the result data frame should have other names, `new_column` can be input as a named vector, see examples.

`position` is used to give the position if the new columns in the data.frame. For `position = "right"` the new variables are placed to the right of the code_variable. Likewise, for `position = "left"` the new variables are placed to the left of the code_variable. If `position = "first"` or `position = "last"` the new columns are placed first or last, respectively, in the data frame. A special case occurs for `position = "keep"` which only has meaning when the new column has the same name as an existing column and `overwrite = TRUE`. In these cases, the existing column will be overwritten with new data and have the same position.

`read_sonetilhorighet` reads the file "sonetilhorighet.txt" into a data frame that can be used by other routines. Standard setting will the file read in the latest updated file from NVI's internal network. If changing the from_path, the function can be used to read the translation file from other directories. This can be useful if having a stand alone app with no connection the NVI's internal network. In other cases, it should be avoided.

Value

`add_lokalitet`: `data.frame` where the aquaculture zone and / or geo-coordinates have been added in the column to the right of the column with the LokNr.

`read_sonetilhorighet`: `data.frame` with "LokNr", aquaculture zone and geo-coordinates. If not changing standard input to the function, the standard file at NVI's internal network is read.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:

# READ TRANSLATION TABLE
# Reading from standard directory at NVI's network
sonetilhorighet <- read_sonetilhorighet()

# ADD AQUACULTURE ZONE
eier_lokalitetnr <- c("10298", "10318", "10735", "10814")
olddata <- as.data.frame(eier_lokalitetnr)

# Add new column with aquculture zone
newdata <- add_lokalitet(olddata,
                         translation_table = sonetilhorighet,
                         code_column = c("eier_lokalitetnr" = "LokNr"),
                         new_column = c("produksjonsomraade" = "sone"),
                         position = "left")

# ADD COORDINATES
eier_lokalitetnr <- c("10298", "10318", "10735", "10814")
olddata <- as.data.frame(eier_lokalitetnr)

# Add new columns with longitude and lattitude
newdata <- add_lokalitet(olddata,
                         translation_table = sonetilhorighet,
                         code_column = c("eier_lokalitetnr" = "LokNr"),
                         new_column = c("longitude" = "Longitude_WGS84",
                                        "latitude" = "Latitude_WGS84"))

## End(Not run)
```

---

add_MT_omrader *Manage translation from komnr to MT-avdeling and MT-region*

Description

Function to add columns with MT_avdelingnr, MT_avdelng (name), MT_regionnr and MT_region (name). In addition there are functions to read and copy the translation tables.

Usage

```
add_MT_omrader(
  data,
  translation_table = komnr_2_MT_omrader,
  code_column = c("komnr"),
  new_column = c("MT_avdelingnr", "MT_avdeling", "MT_regionnr", "MT_region"),
  position = "right",
  overwrite = FALSE
```

```
)

copy_MT_omrader(
  filename = list("komnr_2_MT_avdeling.csv", "MT_omrader.csv"),
  from_path = base::paste0(set_dir_NVI("GrunndataLand"), "FormaterteData/"),
  to_path = NULL
)

read_MT_omrader(
  filename = list("komnr_2_MT_avdeling.csv", "MT_omrader.csv"),
  from_path = base::paste0(set_dir_NVI("GrunndataLand"), "FormaterteData/")
)
```

Arguments

data	Data frame with data with a column with a komnr.
translation_table	Data frame with the table for translating from komnr to MT_areas.
code_column	The column with the coded value. Valid values are one of c("komnr", "MT_avdelingnr", "MT_regionnr"). If the column in data has another name, it can be input as a named vector, see examples.
new_column	The new columns that should be included into the data frame. The new columns can be up to c("MT_avdelingnr", "MT_avdeling", "MT_regionnr", "MT_region") depending on code_column. If the new columns in the result data frame should have other names, new_column can be input as a named vector, see examples.
position	[character(1)] The position for the new columns, can be one of c("first", "left", "right", "last", "keep"). The input can be abbreviated, but must be unique, i.e. c("f", "le", "r", "la", "k"). Defaults to "right".
overwrite	[logical(1)] When the new column(s) already exist, the content in the existing column(s) is replaced by new data if overwrite = TRUE. If the new column(s) already exist and overwrite = FALSE, an error is issued. Defaults to FALSE.
filename	a list with the filenames of the source files with the tables for generating the translation table.
from_path	Path for the source files for the translation table.
to_path	Path to where the source files for the translation table should be copied.

Details

add_MT_omrader can be used to translate the komnr into MT_avdelingnr, MT_avdeling, MT_regionnr and MT_region. The function can also be used to translate MT_avdelingnr into MT_avdeling, MT_regionnr and MT_region or to translate MT_regionnr into MT_region. When the code_column in the dataframe is not equal to one of c("komnr", "MT_avdelingnr", "MT_regionnr") the code_column can be input as a named vector. Likewise, if the new columns should be given other names than c("MT_avdelingnr", "MT_avdeling", "MT_regionnr", "MT_region"), the new_column can be input as a named vector, see examples.

add_MT_omrader uses a premade translation table ("komnr_2_MT_avdeling.csv"). These data need to be loaded by read_MT_omrader before running add_MT_omrader, see example. "komnr_2_MT_avdeling.csv" is made based on information in PJS adresseregister. The translation table is updated when we know there is a need.

position is used to give the place if the new columns in the data frame. For position = "right" the new variables are placed to the right of the code_variable. Likewise, for position = "left" the new variables are placed to the left of the code_variable. If position = "first" or position = "last" the new columns are placed first or last, respectively, in the data.frame. A special case occurs for position = "keep" which only has meaning when the new column has the same name as an existing column and overwrite = TRUE. In these cases, the existing column will be overwritten with new data and have the same position.

read_MT_omrader reads the files "komnr_2_MT_avdeling.csv" and "MT_omrader.csv" into a data frame, usually named komnr_2_MT_omrader. This file is used by add_MT_omrader. If no options to the function is given, the function will read the latest updated files from NVI's internal network. If changing the from_path, the function can be used to read the translation file from other directories. This can be useful if having a script that don't have access to NVI's internal network.

copy_MT_omrader Copies the csv-files "komnr_2_MT_avdeling.csv" and "MT_omrader.csv" to another directory. Thereby, these files are available for read_MT_omrader if they should be read from another directory.

Value

add_MT_omrader A data frame where the MT_avdelingnr has been added in the column to the right of the column with the komnr.

read_MT_omrader A data frame with the table for translating from komnr to c("MT_avdelingnr", "MT_avdeling", "MT_regionnr", "MT_region") as read from the source csv file. If not changing standard input to the function, the standard files at NVI's internal network is read.

copy_MT_omrader Copies the csv-files "komnr_2_MT_avdeling.csv" and "MT_omrader.csv" to another directory. If the target files already exists the source files are only copied if they are newer than the target files.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Reading from standard directory at NVI's network
komnr_2_MT_omrader <- read_MT_omrader()

# Copy the csv-files used to generate the translation table from the standard location to
# the subdirectory Data below the working directory
copy_MT_omrader(to_path = "./Data/")

# Reading from the subdirectory Data below the working directory
komnr_2_MT_omrader <- read_MT_omrader(from_path = "./Data/")

# Add new columns with MT_avdelingnr, MT_avdeling, MT_regionnr, and MT_region based on komnr
# Remember to load "komnr_2_MT_omrader" by "read_MT_omrader()" before running "add_MT_omrader",
# see above.
newdata <- add_MT_omrader(olddata,
                          translation_table = list(komnr_2_MT_omrader),
                          code_column = "komnr",
                 new_column = c("MT_avdelingnr", "MT_avdeling", "MT_regionnr", "MT_region"))

# Add new columns with MT_avdelingnr and MT_avdeling based on komnr. The colname of the column
# with komnr is komnr and the new columns are renamed to MT_avdnr and MT_avd.
```

```
# Remember to load "komnr_2_MT_omrader" by "read_MT_omrader()" before running "add_MT_omrader",
# see above.
newdata <- add_MT_omrader(olddata,
                          translation_table = list(komnr_2_MT_omrader),
                          code_column = c("komnr" = "komnr"),
                      new_column = c("MT_avdnr" = "MT_avdelingnr", "MT_avd" = "MT_avdeling"))

# Add new columns with MT_region based on MT_regionnr. MT_region is renamed to MT_regionnavn
# Remember to load "komnr_2_MT_omrader" by "read_MT_omrader()" before running "add_MT_omrader",
# see above.
newdata <- add_MT_omrader(olddata,
                          translation_table = list(komnr_2_MT_omrader),
                          code_column = "MT_region",
                          new_column = c("MT_regionnavn" = "MT_region"))

## End(Not run)
```

---

add_PJS_code_description

Manage translation of PJS codes to descriptive text

Description

Functions to adds a column with descriptive text for a column with PJS codes in a data frame with PJS data. You may also use backwards translation from descriptive text to PJS code. In addition there are functions to read and copy an updated version of the PJS code registers.

Usage

```
add_PJS_code_description(
  data,
  translation_table = PJS_codes_2_text,
  PJS_variable_type,
  code_colname,
  new_column,
  position = "right",
  overwrite = FALSE,
  backward = FALSE,
  impute_old_when_missing = FALSE
)

copy_PJS_codes_2_text(
  filename = "PJS_codes_2_text.csv",
  from_path = paste0(set_dir_NVI("Provedata_Rapportering"), "FormaterteData/"),
  to_path = NULL
)

read_PJS_codes_2_text(
  filename = "PJS_codes_2_text.csv",
  from_path = paste0(set_dir_NVI("Provedata_Rapportering"), "FormaterteData/")
)
```

Arguments

data	[data.frame] PJS data with at least one column that have codes for a PJS variable.
translation_table	[data.frame] Table with the code and the description for PJS variables. Defaults to "PJS_codes_2_text".
PJS_variable_type	[character] One or more PJS variables, for example "hensikt". See details for a list of all PJS variables included in the pre made translation table "pjscode_2_descriptions.csv". If more than one code type should be translated, they can be given in the vector. You may also use argument PJS_variable_type = "auto", if code_colname have standardized PJS column names only, see details.
code_colname	[character] The name of the column with codes that should be translated. If several codes should be translated, a vector with the names of the coded variables should be given.
new_column	[character] The name of the new column with the text describing the code. If several codes should be translated, a vector with the new column names should be given. You may also use argument new_column = "auto", if code_colname have standardized PJS column names only, see details.
position	[character] Position for the new columns, can be one of c("first", "left", "right", "last", "keep"). If several codes should be translated, either one value to be applied for all may be given or a vector with specified position for each code to be translated should be given. Defaults to "right".
overwrite	[logical(1)] When the new column(s) already exist, the content in the existing column(s) is replaced by new data if overwrite = TRUE. If the new column(s) already exist and overwrite = FALSE, an error is issued. Defaults to FALSE.
backward	[logical(1)] If TRUE, it translates from descriptive text and back to PJS code, see details. Defaults to FALSE.
impute_old_when_missing	[logical(1)] Should existing value be transferred if no value for the code is found? Defaults to FALSE.
filename	[character(1)] File name of the source file for the translation table for PJS codes.
from_path	[character(1)] Path for the source translation table for PJS codes.
to_path	[character(1)] Path for the target translation table for PJS codes when copying the translation table.

Details

Export of data from PJS will produce data frames in which many columns have coded data. These need to be translated into descriptive text to increase readability.

`add_PJS_code_description` can be used to translate the codes into descriptive text. In a data frame with coded values, the function can return a data frame with the descriptive text in a new column. As default, the descriptive text is input in a new column to the right of the column with codes.

`add_PJS_code_description` uses the pre made translation table "PJS_codes_2_text.csv". The data need to be loaded by `read_PJS_codes_2_text` before running `add_PJS_code_description`, see example. The file "PJS_codes_2_text.csv" is normally updated every night from PJS.

Currently, the translation table has PJS codes and the corresponding description for the PJS variable types given in the first column in the table below. The standardized PJS column name is given in the column "code colname" for which the "PJS variable type" will translate into descriptive text. The standard new column name is given in the column "new column".

PJS variable type	code colname	new column	remark
seksjon	ansvarlig_seksjon	ansvarlig_seksjon_navn	
seksjon	utf_seksjon	utforende_seksjon_navn	
hensikt	hensiktkode	hensikt	
utbrudd	utbruddnr	utbrudd	translates NVI's outbreak number
registertype	rekvirenttype	rekvirenttype_navn	categories of locations and addresses
registertype	eier_lokalitettype	eier_lokalitettype_navn	categories of locations and addresses
registertype	annen_aktortype	annen_aktortype_navn	categories of locations and addresses
art	artkode	art	species and breed codes to species name
artrase	artkode	art	species and breed codes to species or breed
fysiologisk_stadium	fysiologisk_stadiumkode	fysiologisk_stadium	
kjonn	kjonn	kjonn_navn	
driftsform	driftsformkode	driftsform	
oppstalling	oppstallingkode	oppstalling	
provetype	provetypekode	provetype	
provemateriale	provematerialekode	provemateriale	
forbehandling	forbehandlingkode	forbehandling	
metode	metodekode	metode	
metode	subund_metodekode	submetode	
konkl_type	konkl_typekode	konkl_type	
kjennelse	sakskonkl_kjennelsekode	sakskonkl_kjennelse	
kjennelse	konkl_kjennelsekode	konkl_kjennelse	
kjennelse	res_kjennelsekode	res_kjennelse	
kjennelse	subres_kjennelsekode	subres_kjennelse	
analytt	sakskonkl_analyttkode	sakskonkl_analytt	
analytt	konkl_analyttkode	konkl_analytt	
analytt	res_analyttkode	res_analytt	
analytt	subres_analyttkode	subres_analytt	
enhet	enhetkode	enhet	
enhet	subres_enhetkode	subres_enhet	

If `code_colname` is a vector of standardized PJS column names and a subset of "code column" in the table above, you may facilitate coding by setting `PJS_variable_type = "auto"` and/or `new_colname = "auto"`. Then the `PJS_variable_type` will be automatically set according to the table above (for "artkode" `PJS_variable_type = "art"` will be chosen). Likewise, the `new_column` will be automatically set according to the table above.

`position` is used to give the position if the new columns in the data frame. For `position = "right"` the new variables are placed to the right of the code_variable. Likewise, for `position`

= "left" the new variables are placed to the left of the code_variable. If position = "first" or position = "last" the new columns are placed first or last, respectively, in the data frame. A special case occurs for position = "keep" which only has meaning when the new column has the same name as an existing column and overwrite = TRUE. In these cases, the existing column will be overwritten with new data and have the same position.

backward = TRUE can be used to translate from descriptive text and back to PJS codes. This intended for cases where the PJS code has been lost (for example in EOS data) or when data from other sources should be translated to codes to be able to use the code hierarchy for further processing of the data. Back translation ignores case. Be aware that the back translation is most useful for short descriptive text strings, as longer strings may have been shortened and the risk of misspelling and encoding problems is larger. For some descriptive text strings, there are no unique translation. In these cases, the code value is left empty.

read_PJS_codes_2_text reads the file "PJS_codes_2_text.csv" into a data frame that can be used by add_PJS_code_description. In standard setting will the file read in the latest updated file from NVI's internal network. If changing the from_path, the function can be used to read the translation file from other directories. This can be useful if having a stand alone app with no connection the NVI's internal network. In other cases, it should be avoided.

"PJS_codes_2_text.csv" has the following columns: c("type", "kode", "navn", "utgatt_dato"), where "type" is the PJS variable type as listed above (for example hensikt), "kode" is the variable with the PJS code, "navn" is the text describing the code, and "utgatt_dato" is the date for last date that the code was valid (NA if still valid). If translation tables are needed for other PJS variables, a data frame with the same column definition can be constructed to translate new variables.

copy_PJS_codes_2_text copies the file "pjsCodeDescriptions.csv" to a given directory.

Value

add_PJS_code_description A data frame where the description text for the PJS code has been added in the column to the right of the column with the code. If the input is a tibble, it will be transformed to a data frame.

read_PJS_codes_2_text A data frame with the translation table for PJS codes as read from the source csv-file. If not changing standard input, the standard file at NVI's internal network is read.

copy_PJS_codes_2_text Copies the source translation table for PJS codes to another location. If the target file already exists the source file is only copied if it is newer than the target file.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Reading from standard directory at NVI's network
PJS_codes_2_text <- read_PJS_codes_2_text()

# Copy standard file from standard location to the subdirectory Data below the working directory
copy_PJS_codes_2_text(to_path = "./Data/")

# Reading from the subdirectory Data below the working directory
PJS_codes_2_text <- read_PJS_codes_2_text("PJS_codes_2_text.csv", "./Data/")

# Translating artkode into art
newdata <- add_PJS_code_description(olddata, PJS_codes_2_text, "art", "artkode", "art")
```

```
# Translating hensiktkode into Hensikt and konklusjonkode to Konklusjonskjennelse
newdata2 <- add_PJS_code_description(olddata,
                                     PJS_codes_2_text,
                                     PJS_variable_type = c("hensikt", "kjennelse"),
                                     code_colname = c("hensiktkode", "konklusjonkode"),
                                     new_column = c("hensikt", "konklusjonskjennelse"))

# Translating hensiktkode into hensikt and konklusjonkode to konklusjonskjennelse using "auto"
newdata3 <- add_PJS_code_description(olddata,
                                     PJS_codes_2_text,
                                     PJS_variable_type = c("auto"),
                               code_colname = c("artkode", "hensiktkode", "konklusjonkode"),
                                     new_column = c("auto"))

# Translating art with species and breed names to only species names
# First the text in art is back-translated to the artkode
newdata4 <- add_PJS_code_description(data = olddata,
                                     PJS_variable_type = "artrase",
                                     code_colname = "art",
                                     new_column = "artkode",
                                     backward = TRUE,
                                     impute_old_when_missing = TRUE)

# Thereafter, the code is translated to art
# By using `impute_old_when_missing = TRUE`, you ensure that text that cannot
# be translated back to code, is reported as text in the end result.
newdata4 <- add_PJS_code_description(data = newdata4,
                                     PJS_variable_type = "art",
                                     code_colname = "artkode",
                                     new_column = "art",
                                     position = "keep",
                                     overwrite = TRUE,
                                     impute_old_when_missing = TRUE)

## End(Not run)
```

---

add_poststed *Manage translation from postnr to poststed and komnr*

Description

Function to add columns with poststed and komnr. In addition there are functions to read and copy the translation tables.

Usage

```
add_poststed(
  data,
  translation_table = poststed,
  code_column = c("postnr"),
  new_column = c("poststed", "komnr"),
  position = "right",
```

```
  overwrite = FALSE
)

copy_poststed(
  filename = "Poststed_UTF8.csv",
  from_path = paste0(set_dir_NVI("GrunndataLand"), "FormaterteData/"),
  to_path = NULL
)

read_poststed(
  filename = "Poststed_UTF8.csv",
  from_path = paste0(set_dir_NVI("GrunndataLand"), "FormaterteData/")
)
```

Arguments

data	Data frame with data with a column with postnr
translation_table	Data frame with the translation table for postnr to poststed and komnr
code_column	The name of the column with the postnr
new_column	The name of the new column that should contain the poststed and/or komnr
position	[character(1)] The position for the new columns, can be one of c("first", "left", "right", "last", "keep"). The input can be abbreviated, but must be unique, i.e. c("f", "le", "r", "la", "k"). Defaults to "right".
overwrite	[logical(1)] When the new column(s) already exist, the content in the existing column(s) is replaced by new data if overwrite = TRUE. If the new column(s) already exist and overwrite = FALSE, an error is issued. Defaults to FALSE.
filename	Filename of the translation table for postnr to poststed and komnr
from_path	Path for the source translation table
to_path	Path for the target translation table when copying the translation table

Details

Data sources may provide data with postnr. These functions manage translating postnr to poststed and komnr.

add_poststed can be used to translate postnr to poststed and komnr.

One has to ensure that the code in the data column represents a postnr. The function will translate any 4 digits that has the same ID as a postnr.

Standard name for the postnr is postnr. If the column with the postnr that should be translated has another name, the parameter code_column can be input as a named vector. Standard names for the new columns are c("poststed", "komnr"). Likewise, if the new columns should be given other names than these, the parameter new_column can be input as a named vector, see examples.

add_poststed uses a premade translation table ("Poststed_UTF8.csv"). These data need to be loaded by read_poststed before running add_poststed, see example. "Poststed_UTF8.csv" is made based on information in PJS adresseregister. The translation table is updated when we know there is a need.

position is used to give the place if the new columns in the data.frame. For position = "right" the new variables are placed to the right of the code_variable. Likewise, for position = "left" the new variables are placed to the left of the code_variable. If position = "first" or position = "last" the new columns are placed first or last, respectively, in the data.frame. A special case occurs for position = "keep" which only has meaning when the new column has the same name as an existing column and overwrite = TRUE. In these cases, the existing column will be overwritten with new data and have the same position.

read_poststed read the file "Poststed_UTF8.csv" a data frame that can be used by add_poststed. Standard setting will read the file from NVI's internal network. If changing the from_path, the function can be used to read the translation files from other directories. This can be useful if having a stand alone app with no connection the NVI's internal network. In other cases, it should be avoided.

copy_poststed copy the file "Poststed_UTF8.csv" to a given directory.

Value

add_poststed A data frame where one or more of the columns c("poststed", "komnr") have been added in the column(s) to the right of the column with the postnr.

read_poststed A data frame with the original postnr and the corresponding poststed and komnr. If not changing standard input to the function, the standard file at NVI's internal network is read.

copy_poststed copies the source translation table for postnr to poststed and komnr to given directory. If the target file already exists, the source file is copied only when it is newer than the target file.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Reading from standard directory at NVI's network
poststed <- read_poststed()

# Copy standard file from standard location to the subdirectory Data below the working directory
copy_poststed(to_path = "./Data/")

# Reading from the subdirectory Data below the working directory
poststed <- read_poststed(from_path = "./Data/")

# Add new column with poststed and komnr
# The variable postnummer should be translated into poststed and komnr. For poststed
# the standard name is kept. For komnr the new variable is named postkomnr.
# Remember to load "poststed" by "read_poststed()" before running "add_poststed",
# see above.
newdata <- add_poststed(olddata,
                        translation_table = poststed,
                        code_column = c("postnummer" = "postnr"),
                        new_column = c("poststed", "postkomnr" = "komnr"))

## End(Not run)
```

add_produsent_properties *Manage translation from prodnr8 into different produsent properties*

Description

Function to add a column with gjeldende_prodnr8. In addition there are functions to read and copy the translation tables.

Usage

```
add_produsent_properties(
  data,
  translation_table,
  code_column,
  new_column,
  position = "right",
  overwrite = FALSE,
  impute_old_when_missing = FALSE
)

copy_prodnr_2_current_prodnr(
  filename = "Prodnr2GjeldendeProdnr.csv",
  from_path = paste0(set_dir_NVI("Prodregister"), "FormaterteData/"),
  to_path = NULL
)

read_prodnr_2_coordinates(
  filename = "Prodnr2Koordinater.csv",
  from_path = paste0(set_dir_NVI("Prodregister"), "FormaterteData/")
)

read_prodnr_2_current_prodnr(
  filename = "Prodnr2GjeldendeProdnr.csv",
  from_path = paste0(set_dir_NVI("Prodregister"), "FormaterteData/")
)
```

Arguments

data	Data frame with data with a column with a prodnr8
translation_table	Data frame with the table for translating from prodnr8 to gjeldende_prodnr8.
code_column	The column with the coded value. Valid values are one of c("prodnr8"). If the column in data has another name, it can be input as a named vector, see examples.
new_column	The new columns that should be included into the data frame. The new columns can be up to c("gjeldende_prodnr8") depending on code_column. If the new columns in the result data frame should have other names, new_column can be input as a named vector, see examples.

position [character(1)]
The position for the new columns, can be one of c("first", "left", "right", "last", "keep"). The input can be abbreviated, but must be unique, i.e. c("f", "le", "r", "la", "k"). Defaults to "right".

overwrite [logical(1)]
When the new column(s) already exist, the content in the existing column(s) is replaced by new data if overwrite = TRUE. If the new column(s) already exist and overwrite = FALSE, an error is issued. Defaults to FALSE.

impute_old_when_missing
Should the ID-variable be used as value for the new_column if the new_column value is missing? Default is FALSE. To be used for translating prodnr8 to gjeldende_prodnr8, see details.

filename a list with the filenames of the source files with the tables for generating the translation table.

from_path Path for the source files for the translation table.

to_path Path for the target translation table when copying the translation table.

Details

add_produsent_properties can be used to translate the prodnr8 into gjeldende_prodnr8 and/or geo-coordinates.

position is used to give the place if the new columns in the data.frame. For position = "right" the new variables are placed to the right of the code_variable. Likewise, for position = "left" the new variables are placed to the left of the code_variable. If position = "first" or position = "last" the new columns are placed first or last, respectively, in the data frame. A special case occurs for position = "keep" which only has meaning when the new column has the same name as an existing column and overwrite = TRUE. In these cases, the existing column will be overwritten with new data and have the same position.

impute_old_when_missing = TRUE is used to replace missing values in the new_column with the value in code_column. This is useful when translating prodnr8 to gjeldende_prodnr8. It should not be used when translating from prodnr8 to something where imputing the old prodnr8 in the new variables don't have any meaning, for example geo-coordinates.

read_prodnr_2_current_prodnr reads the file "Prodnr2GjeldendeProdnr.csv" into a data frame that can be used by other routines. Standard setting will the file read in the latest updated file from NVI's internal network. If changing the from_path, the function can be used to read the translation file from other directories. This can be useful if having a stand alone app with no connection the NVI's internal network. In other cases, it should be avoided.

copy_prodnr_2_current_prodnr copies the file "Prodnr2GjeldendeProdnr.csv" to a chosen directory.

read_prodnr_2_coodinates reads the file "Prodnr2Koordinater.csv" into a data frame that can be used to merge with data frames with prodnr8. Standard setting will the file read in the latest updated file from NVI's internal network. If changing the from_path, the function can be used to read the translation file from other directories. This can be useful if having a stand alone app with no connection the NVI's internal network. In other cases, it should be avoided.

Value

add_produsent_properties returns a data frame where the column with gjeldende_prodnr8 has been added to the right of the column with prodnr8.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# CURRENT PRODNR8
# Reading from standard directory at NVI's network
prodnr_2_gjeldende_prodnr <- read_prodnr_2_current_prodnr()

# Copy standard file from standard location to the subdirectory Data below the working directory
copy_prodnr_2_current_prodnr(to_path = "./Data/")

# Reading from the subdirectory Data below the working directory
prodnr_2_gjeldende_prodnr <- read_prodnr_2_current_prodnr(from_path = "./Data/")

prodnr8 <- c("09140087", "14260856", "17020818", "50060129")
olddata <- as.data.frame(prodnr8)

# Add new column with current prodnr8
newdata <- add_produsent_properties(olddata,
                                    translation_table = prodnr_2_gjeldende_prodnr,
                                    code_column = "prodnr8",
                                    new_column = "gjeldende_prodnr8",
                                    position = "left",
                                    impute_old_when_missing = TRUE)

# COORDINATES
# Reading from standard directory at NVI's network
prodnr_2_koordinater <- read_prodnr_2_coordinates()

newdata <- add_produsent_properties(newdata,
                                    translation_table = prodnr_2_koordinater,
                                    code_column = "prodnr8",
                                    new_column = c("longitude" = "geo_eu89_o",
                                                   "latitude" = "geo_eu89_n"))

## End(Not run)
```

---

build_query_hensikt *Builds query for selecting data for hensikt from PJS*

Description

Builds the query for selecting all data for one or more hensikt within one year from PJS. The query is written in T-SQL as used by MS-SQL.

Usage

```
build_query_hensikt(year, hensikt, db = "PJS")
```

Arguments

year	[numeric] One year or a vector giving the first and last years that should be selected.
hensikt	[character] Vector with one or more specific hensiktkoder. If sub-hensikter should be included, end the code with %.
db	[character(1)] The database for which the query is built. Defaults to "PJS" that currently is the only valid value.

Details

The function builds the SQL syntax to select all PJS-journals concerning the hensiktkoder from PJS.

Value

A list with select-statements for "v2_sak_m_res" and "v_sakskonklusjon", respectively. The statements can thereafter be included in a `RODBC::sqlQuery`.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
# SQL-select query for Pancreatic disease (PD)
build_query_hensikt(year = 2020,
                    hensikt = c("0200102"))
```

---

build_query_one_disease

Builds query for selecting data for one disease from PJS

Description

Builds the query for selecting all data for one infection/disease within one year from PJS. The input is the analytter for the infectious agent and/or disease, the hensikter and metoder specific for the infection and/or disease. The the query is written in T-SQL as used by MS-SQL.

Usage

```
build_query_one_disease(
  year,
  analytt,
  hensikt = NULL,
  metode = NULL,
  db = "PJS"
)
```

Arguments

year	[numeric] One year or a vector giving the first and last years that should be selected.
analytt	[character] Analyttkoder that should be selected. If sub-analytter should be included, end the code with %.
hensikt	[character] Specific hensiktkoder. If sub-hensikter should be included, end the code with %. Defaults to NULL.
metode	[character] Specific metodekoder. Defaults to NULL.
db	[character(1)] The database for which the query is built. Defaults to "PJS" that currently is the only valid value.

Details

The function builds the SQL syntax to select all PJS-journals concerning one infection and/or disease from PJS. This is based on selecting all journals with the disease and/or infectious agent analytt in resultat, konklusjon or sakskonklusjon. By this, all journals were the examination have been performed and a result has been entered should be selected.

One or more specific hensikter may be input to the selection statement. With specific hensikt is meant a hensikt that will imply that the sample will be examined for the infectious agent or disease. Thereby, the selection will include samples that haven't been set up for examination yet, samples that were unfit for examination and samples for which wrong conclusions have been entered.

One or more specific metoder may be input to the selection statement. With specific metode is meant a metode that implies an examination that will give one of the input analytter as a result. Thereby, the query will include samples that have been set up for examination, but haven't been examined yet, samples that were unfit for examination and samples for which wrong results have been entered.

To select both the disease analytt and the infectious agent analytt ensures that all journals that have been examined with a result is included in the output. The inclusion of specific hensikter and metoder, if exists, ensures that all journals received with the purpose of examining for the infectious agent and/or disease will be included even if the examination has not been performed. This is important for a full control of all relevant data for an infectious agent and/or disease.

Value

A list with select-statements for "v2_sak_m_res" and "v_sakskonklusjon", respectively. The statements can thereafter be included in a RODBC::sqlQuery.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
# SQL-select query for Pancreatic disease (PD)
build_query_one_disease(year = 2020,
                        analytt = c("01220104%", "1502010235"),
                       hensikt = c("0100108018", "0100109003", "0100111003", "0800109"),
                        metode = c("070070", "070231", "010057", "060265"))
```

`build_query_outbreak` *Builds query to select data for a disease outbreak from PJS*

Description

Builds a query to select all data for a disease outbreak from PJS. The input are utbruddsid, hensiktskoder, analyttkoder for the infectious agent and/or disease, and metodekoder specific for the infection and/or disease. The the query is written in T-SQL as used by MS-SQL.

Usage

```
build_query_outbreak(
  period,
  utbrudd = NULL,
  hensikt = NULL,
  analytt = NULL,
  metode = NULL,
  db = "PJS"
)
```

Arguments

`period`	[`numeric`] Time period given as year. One year or a vector giving the first and last years that should be selected.
`utbrudd`	[`character`] Utbruddsid(er) that should be selected. Defaults to `NULL`.
`hensikt`	[`character`] Specific hensiktkoder. If sub-hensikter should be included, end the code with %. Defaults to `NULL`.
`analytt`	[`character`] Analyttkoder that should be selected. If sub-analytter should be included, end the code with %. Defaults to `NULL`.
`metode`	[`character`] Specific metodekoder. Defaults to `NULL`.
`db`	[`character(1)`] The database for which the query is built. Defaults to "PJS" that currently is the only valid value.

Details

The function builds the SQL syntax to select all PJS-saker regarding a disease outbreak from PJS. This is based on a outbreak being defined by an utbruddsid, hensiktkoder and/or analyttkoder for the infectious agent and/or disease. At least one of these must be given as input to the function.

The utbruddsid is the internal id of the utbrudd in the utbrudds-register in PJS. One or more utbruddsid may be given as input.

One or more hensiktkoder may be input to the selection statement. These may define the outbreak by themselves or may be input in addition to the utbruddsid and/or analyttkode.

One or more analyttkoder may be input to the selection statement. These may define the outbreak by themselves or may be input in addition to the utbruddsid and/or hensiktkode.

In addition one or more specific metoder may be input to the selection statement. With specific metode is meant a metode that implies an examination that will give one of the input analytter as a result. These cannot be sufficient to define the outbreak, but is included if the outbreak is defined as all samples examined for a specific analytt.

Value

A list with select-statements for "v2_sak_m_res" and "v_sakskonklusjon", respectively. The statements can thereafter be included in a `RODBC::sqlQuery`.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
# SQL-select query for an outbreak
build_query_outbreak(period = 2022,
                     utbrudd = "27",
                     hensikt = c("0100101014", "0100102005", "0100103005",
                                 "0100104029", "0200130%"),
                     analytt = "01130301%",
                     metode = NULL)
```

`build_sql_modules` *Builds sql modules to be included in select statements for PJS*

Description

Builds sql modules to be included in select statements for PJS when building queries for selecting data. The functions takes the values for which observations should be selected as input and builds the sql syntax.

Usage

```
build_sql_select_year(year, varname, db = "PJS")

build_sql_select_code(values, varname, db = "PJS")
```

Arguments

`year`	[`numeric`] One year or a vector giving the first and last years that should be selected.
`varname`	[`character(1)`] The PJS variable name of the variable in PJS from which the coded values should be selected.
`db`	[`character(1)`] The database for which the query is built. Defaults to "PJS" that currently is the only valid value.

values [character]
The value of the codes that should be selected. If sub-codes should be included, add "%" after the code, see example.

Details

build_sql_select_year builds the SQL syntax to select observations from one or more consecutive years from PJS. The input can be given as one year, the first and last year or a range of years. If a range is given, this will be interpreted as first and last years and all years in between will be included.

build_sql_select_code builds the SQL syntax to select observations with the given code values from one variable in PJS with hierarchical codes. When the code value including sub codes should be selected, add " code, see example.

Be aware that these functions only builds an sql building block to be included into a select statement. It will not build a complete select statement. These functions are mainly intended for internal use and are called from build_query_hensikt, build_query_one_disease and build_query_outbreak. If generating own select statements, these can be used to facilitate the coding. The building blocks can be combined with "AND" and "OR" and brackets to get the intended select statement.

Value

SQL-code to be included when building select-statements for PJS.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
# SQL-select module for selecting year from PJS
build_sql_select_year(year = 2020, varname = "aar")

build_sql_select_year(year = c(2019, 2021), varname = "aar")

build_sql_select_year(year = c(2019:2021), varname = "aar")

# SQL-select module for selecting hensiktkode from PJS
build_sql_select_code(values = "0100101", varname = "hensiktkode", db = "PJS")

build_sql_select_code(values = "0100101%", varname = "hensiktkode", db = "PJS")

build_sql_select_code(values = c("0100101", "0100101007", "0100102%", "0100202%"),
                      varname = "hensiktkode",
                      db = "PJS")
```

---

choose_PJS_levels *Choose columns from specified PJS-levels*

Description

Fast way to specify the variables from specific PJS-levels.

Usage

```
choose_PJS_levels(
  data,
  levels,
  keep_col = NULL,
  remove_col = NULL,
  unique_rows = TRUE
)
```

Arguments

data	Data frame with data from PJS
levels	PJS-levels from which data should be chosen. Valid values are c("sak", "prove", "delprove", "undersokelse", "resultat", "konklusjon", "subundersokelse", "subresultat").
keep_col	Column names of columns that should be included in addition to the columns defined by levels.
remove_col	Column names of columns that should be removed even if being at the defined levels.
unique_rows	If TRUE (default), only unique rows are included in the data frame.

Details

When reading PJS-data through certain views, data from more levels that needed may have been read. Some views will also generate so-called Cartesian product increasing the number of rows considerably. By choosing columns from only specified levels the number of unique rows may be reduced considerably.

The function will include columns with colnames that follows the conventional column names as given after using `standardize_columns`. In addition, column names that are the same as the standardized names but without the suffix "kode", will be included into the specified levels.

As standard, only unique (distinct) rows are output. This can be changed by specifying `unique = FALSE`.

Value

A data frame with columns from the chosen levels in PJS.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Attach packages
library(RODBC)
library(NVIdb)

# Read from PJS
journal_rapp <- login_by_credentials_PJS()
PJSdata <- sqlQuery(journal_rapp,
                    "select *
```

```
                    from V2_SAK_M_RES
                  where aar = 2020 and ansvarlig_seksjon = '01' and innsendelsesnummer = 1",
                    as.is = TRUE,
                    stringsAsFactors = FALSE)
odbcClose(journal_rapp)

# Generate two data frames,
#  generates data frame with sak, prove, konklusjon
s_p_k <- choose_PJS_levels(PJSdata,
                           levels = c("sak", "prove", "konklusjon"),
                           remove_col = c("vet_distriktnr", "karantene",
                                          "kartreferanse", "epi_id", "landnr",
                                          "uttatt_parprove", "mottatt_parprove",
                                          "eksportland", "importdato",
                                          "tidl_eier", "avkom_imp_dyr",
                                          "okologisk_drift", "skrottnr",
                                          "kjonn", "fodselsdato", "konklnr"),
                           unique_rows = TRUE)

#  generates data frame with sak, prove, undersokelse and resultat
s_p_u_r <- choose_PJS_levels(PJSdata,
                             levels = c("sak", "prove", "undersokelse", "resultat"),
                             remove_col = c("vet_distriktnr", "karantene",
                                            "kartreferanse", "epi_id", "landnr",
                                            "uttatt_parprove", "mottatt_parprove",
                                            "eksportland", "importdato",
                                            "tidl_eier", "avkom_imp_dyr",
                                            "okologisk_drift", "skrottnr",
                                            "kjonn", "fodselsdato"),
                             unique_rows = TRUE)

## End(Not run)
```

---

copy_Pkode_2_text *Manage translation table for produksjonstilskuddskoder (Pkoder)*

Description

Read and copy the translation table for produksjonstilskuddskoder (Pkoder).

Usage

```
copy_Pkode_2_text(
  filename = "Produksjonstilskuddskoder2_UTF8.csv",
  from_path = paste0(set_dir_NVI("Prodtilskudd"), "StotteData/"),
  to_path = NULL
)

read_Pkode_2_text(
  filename = "Produksjonstilskuddskoder2_UTF8.csv",
  from_path = paste0(set_dir_NVI("Prodtilskudd"), "StotteData/"),
  keep_old_names = FALSE
)
```

Arguments

filename	Name of the translation table, defaults to "Produksjonstilskuddskoder2_UTF8.csv"
from_path	Path for the translation table for produksjonstilskuddskoder
to_path	Path for the target translation table when copying produksjonstilskuddskoder
keep_old_names	[logical(1)] Keep old column names as were used as standard in NVIdb <= v0.7.1. Defaults to FALSE.

Details

The translation table for Pkoder contains the Pkode, descriptive text, unit of interest (Dyr), whether the code counts unique animals or not (used when summarising number of animals), and sortering to order the Pkoder. The register covers 2017 and later.

read_Pkode_2_text reads the file "Produksjonstilskuddskoder2_UTF8.csv" into a data frame. The standard settings will read the file from NVI's internal network. If changing the from_path, the function can be used to read the translation file from other directories. This can be useful if having a stand alone app with no connection the NVI's internal network. In other cases, it should be avoided.

copy_Pkode_2_text copies the file "Produksjonstilskuddskoder2_UTF8.csv" to a given location.

Value

read_Pkode_2_text A data frame with the translation table for Pkoder to description as read from the csv file. If not changing standard input to the function, the standard file at NVI's internal network is read.

copy_Pkode_2_text Copies the source translation table "Produksjonstilskuddskoder2_UTF8.csv" to another location. If the target file already exists, the source file is only copied when it is newer than the target file.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Reading from standard directory at NVI's network
Pkode_2_text <- read_Pkode_2_text()

# Copy standard file from standard location to the sub directory Data below the working directory
copy_Pkode_2_text(to_path = "./Data/")

# Reading from the sub directory Data below the working directory
Pkode_2_text <- read_Pkode_2_text(from_path = "./Data")

## End(Not run)
```

`copy_Prodtilskudd` *Read Register for søknad om produksjonstilskudd*

Description

Functions to to read and copy versions of the produksjonstilskuddsregister.

Usage

```
copy_Prodtilskudd(
  from_path = paste0(set_dir_NVI("Prodtilskudd"), "FormaterteData/"),
  to_path = NULL,
  Pkode_year = "last",
  Pkode_month = "both",
  extracted_date = NULL
)

read_Prodtilskudd(
  from_path = paste0(set_dir_NVI("Prodtilskudd"), "FormaterteData/"),
  Pkode_year = "last",
  Pkode_month = "both",
  extracted_date = NULL
)
```

Arguments

`from_path`	[character(1)] Path for the produksjonstilskuddsregister. Defaults to the standard directory at the NVI network.
`to_path`	[character(1)] Target path for the files with the produksjonstilskuddsregister.
`Pkode_year`	[character]|[numeric] The year(s) from which the register should be read. Options is "last", or a vector with one or more years. Defaults to "last".
`Pkode_month`	[character] The month for which the register should be read. The options are c("05", "10", "both", "last") for `Pkode_year = 2017` and c("03", "10", "both", "last") for `Pkode_year` >= 2018. Defaults to "both".
`extracted_date`	[character] The date the data was extracted from the database of the Norwegian Agricultural Agency. The format should be "yyyy-mm-dd". Defaults to `NULL`.

Details

The produksjonstilskuddsregister includes information on number of animals that the produsent has applied subsidies for at the counting dates. Since 2017, the counting dates are in March and October. Landbruksdirektoratet provides three to four versions of the register for each counting date. The functions automatically selects the last updated version of the register.

`read_Prodtilskudd` reads the produksjonstilskuddsregister into a data frame. The function gives options to select year and season The standard settings will read in the files from NVI's internal

network and select the latest updated file for both spring and autumn and combine them into one file. If changing the from_path, the function can be used to read the translation file from other directories. This can be useful if having a stand alone app with no connection the NVI's internal network. In other cases, it should be avoided.

`extracted_date` is used if specific versions of the register is required, for example to reproduce the generation of data previously performed using an older version of the register.You should also write in the `extracted_date` in the script to document which version of the register that was used. If so, first extract the last available version of the register. Find the uttrekkdato in the data, and write in the uttrekkdato in `extracted_date`. `extracted_date` cannot be used in combination with `pkode_year = "last"` or `pkode_month = c("last", "both")`.

`copy_Prodtilskudd` copies the source produksjonstilskuddsregister for each of the year and seasons selected to a given directory.

Value

`read_Prodtilskudd` reads one or more data frame(s) with the produksjonstilskuddsregister for each of the year and seasons selected. If the options `Pkode_year = "last"` and `Pkode_month = "last"` is given, one file with the last produksjonstilskuddsregister is given.

`copy_Prodtilskudd` copies the source produksjonstilskuddsregister for each of the year and seasons selected. If the target file already exists, the source files are copied only when newer than the target file.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Reading from standard directory at NVI's network
Pkode_last <- read_Prodtilskudd()

# Reading from standard directory at NVI's network and
#     selecting a specific version of the register
Pkode201903 <- read_Prodtilskudd(Pkode_year = "2019", Pkode_month = "03")

## End(Not run)
```

---

cut_slash *Cut away ending slash from string*

Description

Removes ending slash or backslash from string. This is used to clean pathnames so that elements in a path can be combined using `file.path` in stead of `paste0`.

Usage

```
cut_slash(x)
```

Arguments

x	Object with character strings.

Value

Object without ending slash in character strings.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
# Remove from string
cut_slash("C:/temp/")
cut_slash("C:\\temp\\")
cut_slash(c("C:/temp/", "C:\\temp\\"))
cut_slash(list("C:/temp/", "C:\\temp\\"))
```

---

exclude_from_PJSdata *exclude rows from PJS-data*

Description

Performs common subsetting of PJS-data by excluding rows

Usage

```
exclude_from_PJSdata(PJSdata, abroad = "exclude", quality = "exclude")
```

Arguments

PJSdata	Data frame with data extracted from PJS.
abroad	If equal "exclude", samples from abroad are excluded. Allowed values are c("exclude", "include").
quality	If equal "exclude", samples registered as quality assurance and ring trials are excluded. Allowed values are c("exclude", "include").

Details

Performs common cleaning of PJSdata by removing samples that usually should not be included when analyzing PJSdata. The cleaning is dependent on having the following columns eier_lokalitettype, eierlokalitetnr and hensiktkode.

`abroad = "exclude"` will exclude samples that have eier_lokalitet of type "land" and eier_lokalitetnr being different from NO. Samples registered on other types than LAND are not excluded.

`quality = "exclude"` will exclude all samples registered s quality assurance and ring trials, i.e. hensiktkode starting with "09".

Value

data frame without excluded PJS-data.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# cleaning sak_m_res
 sak_m_res <- exclude_from_PJSdata(PJSdata = sak_m_res,
                                   abroad = "exclude",
                                   quality = "exclude")

## End(Not run)
```

`login` *Log in to data base services*

Description

Log in to NVI's data base services, in particular journal_rapp/PJS.

Usage

```
login(
  dbservice,
  dbdriver = NULL,
  db = NULL,
  dbserver = NULL,
  dbport = NULL,
  dbprotocol = NULL,
  dbinterface = NULL
)

login_PJS(dbinterface = NULL, ...)

login_by_credentials(
  dbservice,
  dbdriver = NULL,
  db = NULL,
  dbserver = NULL,
  dbport = NULL,
  dbprotocol = NULL,
  dbinterface = NULL
)

login_by_credentials_PJS(dbinterface = NULL, ...)

login_by_input(
  dbservice,
  dbdriver = NULL,
  db = NULL,
```

```
  dbserver = NULL,
  dbport = NULL,
  dbprotocol = NULL,
  dbinterface = NULL,
  dbtext = NULL
)

login_by_input_PJS(dbinterface = NULL, ...)
```

Arguments

dbservice	[character(1)] Name of the database service, for example "PJS" or "EOS". For database services where one don't use the premade wrappers, the name can be chosen freely, but must be the same as used in set_credentials.
dbdriver	[character(1)] Name of database engine. Defaults to NULL.
db	[character(1)] Name of database. Defaults to NULL.
dbserver	[character(1)] Name of database server. Defaults to NULL.
dbport	[character(1)] Port. Defaults to NULL.
dbprotocol	[character(1)] Protocol to be used. Defaults to NULL.
dbinterface	[character(1)] The R-package that is used for interface towards the data base. Defaults to NULL.
...	Other arguments to be passed from the wrappers to login_by_credentials or login_by_input.
dbtext	[character(1)] Gives the possibility of showing another name than the dbservice in the windows asking for username and password when using login_by_input. Defaults to NULL.

Details

The NVI has access to several database services. These functions log in to such services. The functions provides methods to either log in using credentials set in the user profile by set_credentials or use input windows for username and password. Thereby the hard coding of username and password can be avoided.

login is a general function for connecting to databases, where all necessary connection parameters like server name and database name of the database must be input. The database provider can inform you on the connection parameters for their database. In the case that one login to a database service for which the connection parameters have been predefined (i.e. PJS, EOS, sea_sites, Fallvilt and Dataflex), it will be sufficient to provide the parameter dbservice, for example dbservice = "EOS".

Depending on whether username and password have been saved in the users profile at the current computer or not, the user is asked to input credentials.

login_by_input is a general function for connecting to databases, where all necessary connection parameters like server name and database name of the database must be input. The database

provider can inform you on the connection parameters for their database. In the case that one login to a database service for which the connection parameters have been predefined (i.e. PJS, EOS, sea_sites, Fallvilt and Dataflex), it will be sufficient to provide the parameter `dbservice`. The user is always asked to input username and password.

`login_by_credentials` is a general function for connecting to databases, where all necessary connection parameters like server name and database name of the database must be input. The database provider can inform you on the connection parameters for their database. In the case that one login to a database service for which the connection parameters have been predefined (i.e. PJS, EOS, sea_sites, Fallvilt and Dataflex), it will be sufficient to provide the parameter `dbservice`. The user is never asked for username and password, and the function can only be used when the credentials previously have been set in the user's profile at the current computer.

`login_PJS`, `login_by_input_PJS`, and `login_by_credentials_PJS` are wrappers for the functions above where the specifications for the database journal_rapp/PJS have been pre set. The user only need to input username and password. In the case that the username and password for journal_rapp/PJS have been stored in the user profile at the current computer, the user is automatically logged in to journal_rapp. If the password is no longer valid, an error occur. If so, the user must update the username and password by `set_credentials_PJS`.

The wrapper functions `login_EOS`, `login_by_input_EOS`, and `login_by_credentials_EOS` have been deprecated.

The login functions returns an open ODBC-channel to the database service. The database can then be queried by using functions in the package used for data base interface. The data base interface must be one of `odbc`, `RODBC` or, `RPostgreSQL`. The default is given in NVIconfig and is `RODBC` for "SQL server" and `RPostgreSQL` for "PostgreSQL".

When the session is finished, the script shall close the ODBC-channel by `odbcClose("myodbcchannel")` or `odbcCloseAll` when using `RODBC`.

Value

An open ODBC-channel to the database service.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

See Also

`set_credentials`

Examples

```
## Not run:
library(RODBC)
journal_rapp <- login_PJS()
# Reads hensiktregistret from PJS
hensikter <- sqlQuery(journal_rapp,
                      "select * from v_hensikt",
                      as.is = TRUE,
                      stringsAsFactors = FALSE)
#
odbcClose(journal_rapp)

## End(Not run)
```

NVIdb *NVIdb: A package to facilitate the use of the Norwegian Veterinary Institute's databases.*

Description

The NVIdb package provides functions to facilitate downloading and processing of data from NVI's databases, in particular PJS and EOS. The package comprises four categories of functions: manage credentials, login, read and copy data from in-house data registers and translation of codes into descriptive text.

Manage credentials

Set and remove credentials (i.e. password and user name) in the user profile at the current machine. These functions makes it possible to connect to database services automatically in scripts while avoiding hard coding of the password and the user name.

Login

These functions use the credentials set by functions for managing credentials to automatically login to database services. If the credentials have not been set, there are also login function for interactive input of credentials in windows when running scripts.

Read, copy and update R-data from in-house data registers

The NVI has copies of several data registers like kommuneregister, fylkesregister, register for søknad om Produksjonstilskudd. The aim of these functions are to make these registers easily accessible for R-scripts to ensure that one always uses the latest version of data. These functions reads the registers from where they are saved at NVI internal file system. If necessary, there are function to copy or update the registers to local directories when local versions are needed for example for shiny applications.

Translate codes into descriptive text

Data often only includes codes that should be translated into the description text. These functions perform the translation for PJS-codes and codes like komnr.

Author(s)

Maintainer: Petter Hopp <Petter.Hopp@vetinst.no>

Authors:

- Johan Åkerstedt <Johan.Akerstedt@vetinst.no>

Other contributors:

- Norwegian Veterinary Institute [copyright holder]

See Also

Useful links:

- https://github.com/NorwegianVeterinaryInstitute/NVIdb
- Report bugs at https://github.com/NorwegianVeterinaryInstitute/NVIdb/issues

PJS_code_description_colname *Data: PJS_code_description_colname, standard column names for description texts for selected code variables in PJS.*

Description

A data frame with the standard variable names (column names) for the code variables in PJS, their corresponding standard name of the column with the descriptive text and a column with the PJS type that will can be used to translate from the code variable to the descriptive text. The column names of the code variable are the standardised column names, i.e. after running `NVIdb::standardize_columns`.

The raw data can be edited in the "./data-raw/generate_PJS_code_description_colname.R". The `PJS_code_description_colname` is used by `NVIdb::add_PJS_code_description` when using the options `PJS_variable_type = "auto"` and/or `new_column = "auto"`.

Usage

```
PJS_code_description_colname
```

Format

A data frame with 3 variables:

code_colname column name for selected code variables in PJS and that have been standardized using `NVIdb::standardize_columns`

type the type of PJS variable as used by `NVIdb::add_PJS_code_description` to translate PJS-codes to description text

new_column The new standard column names for the corresponding code column name in PJS

Source

"./data-raw/generate_PJS_code_description_colname.R" in package `NVIdb`

PJS_levels *Data: Variables per PJS-level.*

Description

A data frame with the variable names (column names) in PJS and their corresponding PJS-level. The column names are the standardized column names, i.e. after running `NVIdb::standardize_columns`. The raw data can be edited in the "./data-raw/PJS_levels.xlsx" and the the code for preparing of the data frame is written in "./data-raw/generate_PJS_levels.R". The `PJS_levels` is used as input for `NVIdb::select_PJS_levels`.

Usage

```
PJS_levels
```

Format

A data frame with 9 variables:

variable column name for variables read from PJS and standardized using `NVIdb::standardize_columns`

sak columns at sak-level are given value 1

prove columns at prove-level are given value 1

delprove columns at delprove-level are given value 1

undersokelse columns at undersokelse-level are given value 1

resultat columns at resultat-level are given value 1

konklusjon columns at konklusjon-level are given value 1

subundersokelse columns at subundersokelse-level are given value 1

subresultat columns at subresultat-level are given value 1

Details

The variables included into a specific level is given the value 1, if not included they are given the value 0. To ensure that information on a specific level an be traced to the correct sak, all index variables are given value 1.

Source

"./data-raw/PJS_levels.xlsx" in package `NVIdb`

`read_eos_data` *Read EOS data from RaData*

Description

Reads EOS data from RaData. Includes historical data if these exists. It is possible to limit the data to one or more years.

Usage

```
read_eos_data(
  eos_table,
  from_path = paste0(set_dir_NVI("EOS"), "RaData"),
  year = NULL,
  colClasses = "character",
  encoding = "UTF-8",
  ...
)
```

Arguments

eos_table	[character(1)] The name of the table with eos raw data.
from_path	[character(1)] Path for raw data from eos data.
year	[character | numeric] The years to be included in the result. Can be both numeric or character. Defaults to NULL, i.e. no selection.
colClasses	[character] The class of the columns, as in utils::read.table. Defaults to "character".
encoding	[character(1)] The encoding, one of c("UTF-8", "latin1"). Defaults to "UTF-8".
...	Other arguments to be passed to data.table::fread.

Details

read_eos_data uses data.table::fread to read the data with the settings showProgress = FALSE and data.table = FALSE. Other arguments can be passed to data.table::fread if necessary.

The eos_table name is the same name as the name as in the EOS data base.

Value

A data frame with data from EOS.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

---

read_leveransereg *Read Leveranseregisteret for slakt*

Description

Functions to read Leveranseregisteret for slakt.

Usage

```
read_leveransereg(
  filename,
  from_path = paste0(set_dir_NVI("LevReg"), "FormaterteData/"),
  ...
)
```

Arguments

filename	The name of the file with Leveranseregisteret
from_path	Path for Leveranseregisteret
...	Other arguments to be passed to data.table::fread.

Details

The Leveranseregisteret for slakt includes information on carcasses delivered to slaughter. The register include identity of the farmer, slaughterhouse, date of slaughter, animal species, category (age group and sex), and weight. For poultry the individual animal is not reported, but number of slaughtered poultry per categories, slaughterhouse and date.

`read_leveransereg` reads the Leveranseregisteret for slakt into a data frame. The standard settings will read in the files from NVI's internal network. If changing the `from_path`, the function can be used to read Leveranseregisteret from other directories. This can be useful if having a stand alone app with no connection the NVI's internal network. In other cases, it should be avoided.

Value

`read_LevReg` A data frame with Leveranseregisteret as in selected csv-file.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Reading from standard directory at NVI's network
LevReg2019 <- read_leveransereg(filename = "LevReg_201901_201912.csv")

## End(Not run)
```

`read_varekode` *Manage translation table for varekoder til leveransregisteret*

Description

Read the translation table for varekoder til leveransregisteret.

Usage

```
read_varekode(
  filename = "varekoder.csv",
  from_path = paste0(set_dir_NVI("LevReg")),
  year = NULL,
  data_source = "formatted"
)
```

Arguments

`filename`	Name of the translation table, defaults to "varekoder.csv". The input is only used when `data_source = "formatted"`.
`from_path`	Path for the translation table for varekoder.
`year`	Year(s) for fetching the varekoderegister.
`data_source`	Reads formatted data or raw data. Default is formatted.

Details

The translation table for varekoder comprises the variables: the leveranseaar, varekode, vare (descriptive text), dyreslag, vareart, dyrekategori, and varekategorikode. The register covers 2016 and later.

`read_varekoder` with the argument `type = "formatted"` reads the formatted "varekoder.csv" into a data frame. The standard settings will read the file from NVI's internal network. If changing the from_path, the function can be used to read the translation file from other directories. This can be useful if having a stand alone app with no connection the NVI's internal network. In other cases, it should be avoided.

`read_varekoder` with the argument `type = "raw"` reads the raw data as supplied from Landbruksdirektoratet into a data frame. Thereafter, these can be used to generate the formatted version. The standard settings will read the file from NVI's internal network and changing the path should be avoided.

Value

`read_varekoder` A data frame with the translation table for varekoder to descriptive text and metadata.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Reading from standard directory at NVI's network
varekoder <- read_varekode()

## End(Not run)
```

---

remove_credentials *Manage username and password (credentials) for database services at NVI*

Description

Save or remove the current user's username and password for the data base services at the Norwegian Veterinary Institute in the the user profile.

Usage

```
remove_credentials(dbservice)

set_credentials(dbservice)

set_credentials_PJS()
```

Arguments

dbservice	[character(1)] Name of the database service, for example "PJS" or "EOS". For database services where one don't use the premade wrappers, the name can be chosen freely, but must be the same as used in login and login_by_credentials.

Details

The Norwegian Veterinary Institute has access to various database services. To simplify the access to the database services when using R, the function set_credentials makes it possible to save the username and password (credentials) in the user profile at the current machine. When the username and password have been saved in the user profile, the functions login or login_by_credentials will automatically log in to the database services without any need of new input of username and password.

The user profile is not copied between computers. Consequently, if a user runs scripts with login on different computers, the credentials have to be saved at each computer separately.

set_credentials(dbservice) is used to set the username and password for a database service. The username and password are input using windows and saved in the users profile at the current computer. When the password for the database service have been changed, set_credentials(dbservice) can be used to update the password.

set_credentials_PJS is a wrapper for set_credentials(dbservice) used to set the username and password for journal_rapp/PJS. Journal_rapp has views to information in PJS and some other internal databases at NVI. The username and password are the same as for PJS. When the password for PJS have been changed, set_credentials_PJS can be used to update the password.

The wrapper function set_credentials_EOS is deprecated. Use set_credentials(dbservice = "EOS") instead.

remove_credentials(dbservice) is used to delete the credentials for a database service from the user's profile.

Value

set_credentials The username and password for a database service are saved in the user profile at the current computer.

remove_credentials The username and password for a database service are deleted from the user profile at the current computer.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

See Also

login and login_by_credentials

Examples

```
## Not run:
set_credentials(dbservice = "dbservice")

set_credentials_PJS()

set_credentials("EOS")
```

```
remove_credentials("PJS")

## End(Not run)
```

`retrieve_PJSdata` *Retrieves data from PJS*

Description

Retrieves and standardises PJS data. `retrieve_PJSdata` is a wrapper for several `NVIdb`-functions and the intention of `retrieve_PJSdata` is to shorten code and to ensure that a standard procedure is followed when retrieving PJS data, see details. It can only be used for retrieving case data from PJS where the columns "aar", "ansvarlig_seksjon" and "innsendelsenr" are included in the columns. It cannot be used for retrieving data from other tables available in "journal_rapp".

Usage

```
retrieve_PJSdata(
  year = NULL,
  selection_parameters = NULL,
  FUN = NULL,
  select_statement = NULL,
  ...
)
```

Arguments

`year`	`[numeric]` One year or a vector giving the first and last years that should be selected. Defaults to `NULL`.
`selection_parameters`	`[character(1)]` Either the path and file name for an R script that can be sourced and that sets the selection parameters or a named list with the selection parameters (i.e. of the same format as the output of `set_disease_parameters`). Defaults to `NULL`.
`FUN`	`deprecated` `FUN` should instead be included as input to `selection_parameters`. Defaults to `NULL`.
`select_statement`	`deprecated` `select_statement` should instead be included as input to `selection_parameters`. Defaults to `NULL`.
`...`	Other arguments to be passed to the underlying functions: `login_PJS` and `exclude_from_PJSdata`.

Details

`retrieve_PJSdata` is a wrapper for the following `NVIdb`-functions:

- Constructs the select statement by a build_query-function (see details) and selection parameters.

- Creates an open ODBC-channel using `login_PJS`.
- Retrieves the data using the select statement constructed above.
- Standardises the data using `standardize_PJSdata`.
- Excludes unwanted cases using `exclude_from_PJSdata`.

For the function to run automatically without having to enter PJS user credentials, it is dependent that PJS user credentials have been saved using `set_credentials_PJS`. Otherwise, the credentials must be input manually to establish an open ODBC channel.

The select statement for PJS can be built giving the selection parameters and input to one of the build_query-functions, i.e. `build_query_hensikt`, `build_query_one_disease` and `build_query_outbreak`. The selection parameters can be set by using `set_disease_parameters`. or by giving a list of similar format for input to `selection_parameters`, see the build_query-functions for necessary input.

`retrieve_PJSdata` gives the possibility of giving the select_statement as a string instead of using the build_query-functions. If so, the select_statement should be included in the selection parameters. This should only by done for select statements that previously have been tested and are known to have correct syntax. `retrieve_PJSdata` has no possibility of checking the sql syntax before it is submitted to PJS and untested select statements can take a lot of time or stop the function without proper error messages. In the case that both a select_statement and a function with the necessary selection_parameters are given, the select_statement constructed by the function will be used.

The output is a named list where each entry is a data frame with PJS data. If the select statement is named, the returned data frame will have that name. If the select statement is unnamed, it will try to identify the first table in the select statement and use this as name. If not possible, the name will be of the format "PJSdata#" where # is the number of the select statement.

Value

A named list with PJS data.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

#

`select_PJSdata_for_value`

Selects a subset of PJSdata based on code values

Description

Selects a subset of PJSdata based on code values. The function accepts code values ending with " that sub levels should be included.

Usage

```
select_PJSdata_for_value(
  data,
  code_column,
  value_2_check,
  keep_selected = TRUE
)
```

Arguments

data	[data.frame] PJS data from which a subset should be selected.
code_column	[character] Vector with the column names for the variables that is used in the selection.
value_2_check	[character] Vector with the values that should be selected, see details and examples.
keep_selected	[logical(1)] If TRUE, the selected rows are included, if FALSE, the selected columns are excluded. Defaults to TRUE.

Details

The function is intended for cases where the select query sent to PJS will be very complicated if the selection is included and it can be easier to read the script if the subset is selected in a second step.

The function selects according to different values. The default action is to include the selected rows. But when keep_selected = FALSE, the selected rows are excluded from the data.

Value

A data.frame.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

---

set_dir_NVI *Set directories for data sources at NVI*

Description

Set the directories for various data sources at NVI's network.

Usage

```
set_dir_NVI(datasource, slash = TRUE)
```

Arguments

datasource	[character(1)] The data source that one want to access. The input can be abbreviated and case is ignored. To identify short names for the available directories, use set_dir_NVI(datasource = "?").
slash	[logical(1)] If TRUE the path ends with a slash. Defaults to TRUE.

Details

The Norwegian Veterinary Institute has standard data sources at fixed directories. The function returns the standard directory for the given data source. Thereby hard coding of the paths may be avoided.

The path ends with a slash as default. To facilitate the use of file.path you can use the argument slash = FALSE to avoid ending slash.

Value

The full path for the directory at NVI's network. The path ends with "/" as default, see details.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Set path_ProdTilskudd to path for Prodtilskudd at the NVI network
prodtilskudd_path <- set_dir_NVI(datasource = "ProdTilskudd")

# Set pathname to a file using 'file.path'
pathname <- file.path(set_dir_NVI(datasource = "ProdTilskudd", slash = FALSE),
                      "subdir",
                      "filename")

## End(Not run)
```

---

set_disease_parameters

Sets disease selection parameters

Description

Sets the disease selection parameters and store them in a list object. The list follows a standardised named format and the elements can be used as input to build_query_hensikt, build_query_one_disease or build_query_outbreak.

Usage

```
set_disease_parameters(
  purpose = NULL,
  hensikt2select = NULL,
  hensikt2delete = NULL,
  utbrudd2select = NULL,
  metode2select = NULL,
  analytt2select = NULL,
  analytt2delete = NULL,
  art2select = NULL,
  include_missing_art = NULL,
  FUN = NULL,
  select_statement = NULL,
  selection_parameters = NULL,
  ...
)
```

Arguments

purpose	[character] A short description of the purpose of the selection, see details. Defaults to NULL.
hensikt2select	[character] Specific "hensiktkoder" for the "analytt" in question. If sub-codes should be included, end the code with %. Defaults to NULL.
hensikt2delete	[character] "hensiktkoder" for which saker should be excluded. If sub-codes should be included, end the code with %. Defaults to NULL.
utbrudd2select	[character(1)] "utbruddsID". Defaults to NULL.
metode2select	[character] Specific "metodekoder" for the "analytt" in question. Defaults to NULL.
analytt2select	[character] "analyttkoder" for the agent and/or disease. If sub-codes should be included, end the code with %. Defaults to NULL.
analytt2delete	[character] Specific "analyttkoder" that should be deleted, see details. If sub-codes should be included, end the code with %. Defaults to NULL.
art2select	[character] "artkoder". If sub-codes should be included, end the code with %. NA can be combined with another "artkode". Defaults to NULL.
include_missing_art	[character(1)] Should missing art be included. Must be one of c("never", "always", "for_selected_hensikt"). If NULL, it is set to "always" when art2select includes NA, else it is set to "never". Defaults to NULL.
FUN	[function] Function to build the selection statement, see retrieve_PJSdata). Defaults to NULL.

`select_statement`
: [character(1)]
A written select statement, see `retrieve_PJSdata`. Defaults to `NULL`.

`selection_parameters`
: [character(1)]
Either the path and file name for an R script that can be sourced and that sets the selection parameters or a named list with the selection parameters (i.e. equal to the output of this function). Defaults to `NULL`.

`...`
: Other arguments to be passed to `set_disease_parameters`.

Details

Saker in PJS that concern one infection / disease can be characterised by the "analytt" (at "konklusjon" and/or "resultat" level), specific "hensikter", a relevant "utbrudds_ID" and/or specific "metoder." These can be used to select saker in PJS and/or to structure and simplify the output from PJS.

The purpose is a short description of purpose of the selection described by the selection parameters for example "ok_svin_virus" that describes the selection parameters for the OK programme for virus in swine. The purpose is also used as part of the file name for selection_parameters, i.e. "purpose_selection_parameters" and in the annual tables for ok_programmer: Kontrolltabeller for yyyy.

One or more specific "hensiktkoder" may be input to the selection statement. With specific "hensiktkode" is meant a "hensiktkode" that will imply that the sample will be examined for specific infectious agent(s) or disease. One or more specific "metodekoder" may be input to the selection statement. With specific "metodekode" is meant a "metodekode" that implies an examination that will give one of the input 2 as a result. If sub-codes of "analyttkode" or "hensiktkode" should be included, end the code with %.

The selection parameters can be input values for dedicated arguments. For input parameters `hensikt2select`, `hensikt2delete`, `utbrudd2select`, `metode2select`, `analytt2select`, `analytt2delete`, `art2select`, `include_missing_art`, `select_statement`, and `FUN`, the input may be given in a source file. This may be handy if the selection will be performed many times. It also gives the possibility of using a for loop that selects PJS-data and performs similar analyses for one disease at a time.

The selection parameter `analytt2delete` is intended for the situation where `analytt2select` includes analytter higher in the hierarchy and there are specific analytter lower in the hierarchy that should not be included. A typical example is the selection of all samples with the analytt Mycobacterium spp and below, but one is only interested in M. tuberculosis complex but not in M. avium.

The possibility of input other arguments are kept to make it possible to use the deprecated arguments `missing_art` and `file`. If these are used, a warning is issued and the input is transferred to `include_missing_art` and `selection_parameters`, respectively.

Value

A named list with selection parameters that can be used to generate SQL selection-statements and facilitate structuring output from PJS.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
# Selection parameters for Pancreatic disease (PD)
selection_parameters <- set_disease_parameters(
  analytt2select = c("01220104%", "1502010235"),
  hensikt2select = c("0100108018", "0100109003", "0100111003", "0800109"),
  metode2select = c("070070", "070231", "010057", "060265")
  )
```

`standardize_columns` *Standardize columns for scripts and reports*

Description

Standardizes column names, labels, column width for variables in external databases.

Usage

```
standardize_columns(
  data,
  dbsource = deparse(substitute(data)),
  standards = NULL,
  property,
  language = "no",
  exclude = FALSE,
  ...
)
```

Arguments

`data`	[`data.frame`	`character(1)`] The data source. If `property = "colclasses"` the path and file name of the csv-file used as data source should be given.
`dbsource`	[`character(1)`] The database that is the source of data. Should be the name of the data source as registered in column_standards table. Defaults to `deparse(substitute(data))`.
`standards`	[`character(1)`] For giving alternative standard tables to column_standards.
`property`	[`character(1)`] Property of the column that should be standardized. Must be one of c("colnames", "colclasses", "collabels", "colwidths_Excel", "colorder"). Defaults to `NULL`.
`language`	[`character(1)`] Language for labels. Must be one of c("no", "en"). Defaults to "no".
`exclude`	[`logical(1)`] Used in combination with `property = "colorder"`. If `TRUE`, all columns with no predefined column order are excluded. Defaults to `FALSE`.
`...`	Other arguments to be passed to `read.csv2` when `property = "colclasses"`.

Details

The standardization table is under development. This function only works when being connected to the NVI network.

Variables in internal and external data sources uses different variable names for the same content. `standardize_columns` standardizes column names for use in scripts. In addition, it standardises column labels and column widths for Excel. Furthermore, input values for the parameter `colClasses` for `read.csv2` and `data.table::fread` can be generated.

`property = "colnames"` will replace the column names in a data frame with standardized column names. All standard column names is snake_case. If no standard name is defined for a variable name, the variable name is translated to snake_case and the national characters c("æ", "ø", "å") are translated to c("ae", "oe", "aa").

`property = "colclasses"` will generate a named vector with the column classes for variables that may not be read correct when importing data from a csv-file. This applies for example to numbers with leading zero that must be imported as character. This vector can be used as a parameter for `colClasses`.

The default `fileEncoding` is assumed to be "UTF-8". If another encoding is needed, one must give an additional argument like `fileEncoding = "latin1"`.

`property = "collabels"` will generate a vector with column labels that can be used to replace the column names in the header of the data table. The column names are not standardised automatically but can be standardised by first using `standardize_columns` with `property = "colname"`. If no standard column label for the column name is defined, the column name as Sentence case is used as column label. If English names are used and no English column label exists, the Norwegian column label is used instead.

`property = "colwidths_Excel"` will generate a vector with column widths for Excel. To be used as input parameter to `openxlsx::setColWidths`. If no standard column width is defined, the Excel standard width of 10.78 is used. Be aware that the generation of column widths are based on the column names. Do not change the column names to labels before the column widths are generated.

`property = "colorder"` will generate a data frame with the column names in a predefined order. The column names should first have been standardised. No standard order will be given unless the dbsource is defined in the column_standards table. If `exclude = FALSE` (the standard) the columns with no predefined order will be moved to the last columns in the same order as they appeared in the original data frame. If `exclude = TRUE` all columns with no predefined order is excluded from the data frame. This option is mainly intended for well defined and worked through routines like making selections lists for the Food Safety Authority. Do not use `exclude = TRUE` unless you are certain that all columns that should be included are defined in the column_standards table for this dbsource. If uncertain, you may first try with `exclude = FALSE` and thereafter compare with `exclude = TRUE` to check if you loose important information.

Value

`property = "colnames"`: A data frame with standard column names.

`property = "colclasses"`: A named vector of column classes to be used as input to functions for reading csv-files, see details.

`property = "collabels"`: A vector with labels for the columns in the data frame.

`property = "colwidths_Excel"`: A vector with column widths for Excel. To be used as input parameter to `openxlsx::setColWidths`.

`property = "colorder"`: A data frame with column names in predefined order. If `exclude = TRUE` only columns with a defined order is included.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Generate data frame to be standardized
df <- cbind("\u00C5r" = 2020, "Hensiktkode" = "01001", komnr = "5001")
colnames(df)

# Standardize column names
df <- standardize_columns(data = df, property = "colnames")
colnames(df)

# Generate vector with standard labels
labels <- standardize_columns(data = df, property = "collabels")
# use the labels as column names
colnames(df) <- labels

# Generate vector with standard column widths for Excel
colwidths <- standardize_columns(data = df, property = "colwidths_Excel")
colwidths

## End(Not run)
```

---

standardize_eos_data *Standardising EOS-data*

Description

Standardising EOS-data. This standardising should always be performed. Otherwise summary numbers can be wrong.

Usage

```
standardize_eos_data(
  data,
  dbsource = deparse(substitute(data)),
  standards = NULL,
  standardize_colnames = TRUE,
  breed_to_species = TRUE,
  adjust_n_examined = TRUE,
  delete_redundant = TRUE,
  ...
)
```

Arguments

data	[data.frame] The data retrieved from EOS.

dbsource [character(1)]
: If specified, this will be used for fetching standard column names by standardize_columns. Defaults to the name of the input data.

standards [data.frame]
: The translation table to standard column names. Defaults to NULL.

standardize_colnames
: [logical(1)]
If TRUE, the column names will be standardised. Defaults to TRUE.

breed_to_species
: [logical(1)]
If TRUE, breed is translated back to species. Defaults to TRUE.

adjust_n_examined
: [logical(1)]
If TRUE, the number of examined samples is adjusted so it is at maximum the number of received samples. Defaults to TRUE.

delete_redundant
: [logical(1)]
If TRUE, redundant variables in the data is deleted. Defaults to TRUE.

... Other arguments to be passed to standardize_columns.

Details

The function performs the following standardising of data extracted from EOS:

- The column names are standardised using standardize_columns.
- Numeric variables are transformed to numbers.
- Datetime variables are transformed to dates.
- Double registrations of a "Sak" due to the municipality being divided between two Food Safety Authority office, are merged into one and for these, the information on Food Safety Authority office is removed.
- Splits saksnr into saksnr and fagnr if saksnr combines both.
- Breed is transformed to species.
- Number of examined samples are corrected so it don't exceed the number of received samples.
- Redundant variables are deleted.

Standardisation of column names may be set to FALSE. This should only be done if the column names have been standardised previously as a new standardisation of column names may give unpredicted results. Remark that all other standardisations are dependent on standard column names, so the function will not work if the data do not have standard column names.

Transformation from breed to species is only performed when species is included in the data. You need to import the translation table for PJS-codes to perform the translation, use PJS_codes_2_text <- read_PJS_codes_2_text().

Correction of number of tested samples is only done when both number of received and number of tested are included in the data.

There are a few reduntant varibles in some data sets. In CWD data both "sist_overfort" and "sist_endret" keeps the same information. "sist_endret" is deleted. In Salmonella and Campylobacter data, "prove_identitet" is always NULL and "prove_id" is NULL for salmonella data and equal ti "id_nr" for Campylobacter data. Both are deleted. Set delete_redundant = FALSE to keep them.

Value

data.frame with standardized EOS-data.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
# Standardizing proveresultat_bse
PJS_codes_2_text <- read_PJS_codes_2_text()
proveresultat_bse <- standardize_eos_data(data = proveresultat_bse)

## End(Not run)
```

---

standardize_PJSdata *Standardizing PJS-data*

Description

Standardizing PJS-data. This standardizing should always be performed. Other functions used for further preparation of PJSdata, like choose_PJS_levels , and exclude_from_PJSdata will not work as intended unless the column names are standardized.

Usage

```
standardize_PJSdata(PJSdata, dbsource = "v2_sak_m_res")
```

Arguments

PJSdata	[data.frame] Data retrieved from PJS.
dbsource	[character(1)] The table that is the source of data. This will be used for fetching standard column names by standardize_columns and should be the name of the data source as registered in the "column_standards" table. Defaults to "v2_sak_m_res".

Details

The function performs the following standardizing of data extracted from PJS:

- The unnecessary columns konkl_provenr and vet_distriktnr are removed.
- The column names are standardized using standardize_columns.
- Numeric variables are transformed to numbers.
- Date variables are transformed to date format.
- Character variables are trimmed for leading and trailing spaces.
- The variables saksnr and, if possible, fagnr are generated.
- Test data, i.e. saker with ansvarlig_seksjon in c("14", "99") are deleted.

Value

`data.frame` with standardized PJS-data.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Johan Åkerstedt Johan.Akerstedt@vetinst.no

Examples

```
## Not run:
# Standardizing sak_m_res
sak_m_res <- standardize_PJSdata(PJSdata = sak_m_res)

## End(Not run)
```

---

transform_code_combinations

Transform combinations of code values into new values

Description

Transforms combinations of code values into new values in a data frame. This is intended for use when only a few code value combinations should be changed and one will avoid building translation tables or code with several `if`, `which` or `case_when` statements. In particularly it was inspired by the need of changing a few code combinations in PJS data when reporting surveillance programmes.

Usage

```
transform_code_combinations(
  data,
  from_values,
  to_values,
  impute_when_missing_from = NULL
)
```

Arguments

`data`	[`data.frame`] Data with code values that should be transformed.
`from_values`	[`list`] List with named vector(s) of code values that should transformed, see details and examples.
`to_values`	[`list`] List with named vector(s) of code values that should be the results of the transformation, see details and examples.
`impute_when_missing_from`	[`character`] Column names for the code variables from which code values should be copied if no transformation is performed. Defaults to the original column names.

Details

The function builds a transformation table based on the input. The `from_values` and the `to_values` give the data to a transformation table, and the `from_columns` and the `to_columns` give the column names for the transformation table.

The `from_values` is a list of one or more vectors. Each vector is named with the column name and represents one column variable with code values. The first entry in each vector constitute one code combination to be transformed, the second entry constitutes the next code combinations.

Likewise, is the `to_values` a list of one or more named vectors. Each vector is named and represents one column variable with code values to which the code combinations in the `from_values` should be transformed. The name of the vector is the name of the columns with the transformed values. The transformed values can be put in the original columns, in which case the transformed combinations will replace the original entries. If the transformed column names don't exist in data, the columns will be added to the data.

If the codes are not transformed, these can be kept in the data. `impute_when_missing_from` gives the column names of the columns from which to impute. Normally this will be the same as the original columns. However, if the number of transformed columns is less than the original columns, it will be necessary to give the columns from which to keep the code.

Value

A `data.frame`.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
library(NVIdb)

#  A code combination of two is tranformed to another code combination of two
  data <- as.data.frame(cbind(
c("Detected", "Detected", "Not detected", NA),
                              c("M. bovis", "M. kansasii", "M. bovis", NA)
))
  colnames(data) <- c("kjennelse", "analytt")

  data <- transform_code_combinations(data = data,
                                      from_values = list("kjennelse" = c("Detected"),
                                                         "analytt" = c("M. kansasii")),
                                      to_values = list("kjennelse" = c("Not detected"),
                                                       "analytt" = c("M. bovis")),
                                    impute_when_missing_from = c("kjennelse", "analytt"))

# two code values to one new variable
data <- as.data.frame(cbind(c("hjort", "rein", "elg", "hjort", NA),
                        c("produksjonsdyr", "ville dyr", "ville dyr", "ville dyr", NA)))
colnames(data) <- c("art", "driftsform")

data <- transform_code_combinations(
             data = data,
             from_values = list("art" = c("hjort", "rein", NA),
                                "driftsform" = c("produksjonsdyr", "ville dyr", NA)),
             to_values = list("art2" = c("oppdrettshjort", "villrein", "ukjent")),
```

```
                impute_when_missing_from = "art")
```

Index

∗ **Log in functions**
    login, 31
∗ **datasets**
    PJS_code_description_colname, 35
    PJS_levels, 35

add_kommune_fylke, 3
add_lokalitet, 5
add_MT_omrader, 7
add_PJS_code_description, 10
add_poststed, 14
add_produsent_properties, 17

build_query_hensikt, 19
build_query_one_disease, 20
build_query_outbreak, 22
build_sql_modules, 23
build_sql_select_code (build_sql_modules), 23
build_sql_select_year (build_sql_modules), 23

choose_PJS_levels, 24
copy_kommune_fylke (add_kommune_fylke), 3
copy_MT_omrader (add_MT_omrader), 7
copy_PJS_codes_2_text (add_PJS_code_description), 10
copy_Pkode_2_text, 26
copy_poststed (add_poststed), 14
copy_prodnr_2_current_prodnr (add_produsent_properties), 17
copy_Prodtilskudd, 28
cut_slash, 29

exclude_from_PJSdata, 30

login, 31, *40*
login_by_credentials, *40*
login_by_credentials (login), 31
login_by_credentials_PJS (login), 31
login_by_input (login), 31
login_by_input_PJS (login), 31
login_PJS (login), 31

NVIdb, 34
NVIdb-package (NVIdb), 34

PJS_code_description_colname, 35
PJS_levels, 35

read_eos_data, 36
read_kommune_fylke (add_kommune_fylke), 3
read_leveransereg, 37
read_MT_omrader (add_MT_omrader), 7
read_PJS_codes_2_text (add_PJS_code_description), 10
read_Pkode_2_text (copy_Pkode_2_text), 26
read_poststed (add_poststed), 14
read_prodnr_2_coordinates (add_produsent_properties), 17
read_prodnr_2_current_prodnr (add_produsent_properties), 17
read_Prodtilskudd (copy_Prodtilskudd), 28
read_sonetilhorighet (add_lokalitet), 5
read_varekode, 38
remove_credentials, 39
retrieve_PJSdata, 41

select_PJSdata_for_value, 42
set_credentials, *32, 33*
set_credentials (remove_credentials), 39
set_credentials_PJS, *33*
set_credentials_PJS (remove_credentials), 39
set_dir_NVI, 43
set_disease_parameters, 44
standardize_columns, 47, *50*
standardize_eos_data, 49
standardize_PJSdata, 51

transform_code_combinations, 52