Package ‘NVIdb’

September 8, 2022

Title Tools to facilitate the use of NVI's databases

Version 0.7.1.9000

Date 2022-##-##

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 stats,
utils,
checkmate,
data.table,
dplyr,
getPass,
keyring,
knitr,
magrittr,
poorman (>= 0.2.3),
remotes,
rlang,
rmarkdown,
RODBC,
R.rsp,
shiny,
snakecase,
svDialogs,
NVIcheckmate (>= 0.4.0),
NVIrpackages

Suggests covr,
devtools,
spelling,
testthat,
tibble,
NVIpackager

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

LazyData true

Encoding UTF-8

Language en-GB

Roxygen list(markdown = FALSE)

RoxygenNote 7.2.1

VignetteBuilder knitr,
R.rsp

R topics documented:

`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 input giving the position for the new columns, can be one of c("first", "left", "right", "last", "keep"). The input can be abbreviated.
`overwrite`	`TRUE` or `FALSE`, default is `FALSE`. 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.
`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 komnr 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 tables 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 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 with data with a column with a aquaculture site number ("LokNr")
`translation_table`	Data frame with the table for translating from loknr to the property in question.

`code_column`	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`	The new columns that should be included into the data frame. The new columns can be one ore 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`	character input giving the position for the new columns, can be one of c("first", "left", "right", "last", "keep"). The input can be abbreviated.
`overwrite`	`TRUE` or `FALSE`, default is `FALSE`. 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.
`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.

Details

`add_lokalitet` can be used to add aquaculture zone to aquaculture sites.

`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_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` A data frame where the aquaculture zone and / or coordinates have been added in the column to the right of the column with the LokNr.

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 input giving the position for the new columns, can be one of c("first", "left", "right", "last", "keep"). The input can be abbreviated.
`overwrite`	`TRUE` or `FALSE`, default is `FALSE`. 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.
`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
)

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 with PJS-data with a column with codes for a PJS-variable.
`translation_table`	Data frame with the code and the description for PJS-variables.
`PJS_variable_type`	A vector with PJS-variables, for example "hensikt". See details for a list of all PJS-variables included in the premade translation table pjscode_2_descriptions.csv. If more than one code should be translated, they can be given in the vector. You may also use "auto", if `code_colname` have standardized PJS names only, see details.
`code_colname`	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`	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 "auto", if `code_colname` have standardized PJS names only, see details.
`position`	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.
`overwrite`	`TRUE` or `FALSE`, default is `FALSE`. 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.
`backward`	`[logical]`. If `TRUE`, it translate from descriptive text and back to PJS-code, see details. Defaults to `FALSE`.
`filename`	File name of the source file for the translation table for PJS-codes.
`from_path`	Path for the source translation table for PJS-codes.
`to_path`	Path for the target translation table for PJS-codes when copying the 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 a premade translation table (PJS_codes_2_text.csv). The data need to be loaded by `read_PJS_code_registers` 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	
konkl_type	konkl_typekode	konkl_type	

kjennelse	konkl_kjennelsekode	konkl_kjennelse
kjennelse	res_kjennelsekode	res_kjennelse
analytt	konkl_analyttkode	konkl_analytt
analytt	res_analyttkode	res_analytt
enhet	enhetkode	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_code_registers` 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_code_registers` 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"))

## 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 input giving the position for the new columns, can be one of c("first", "left", "right", "last", "keep"). The input can be abbreviated.
`overwrite`	`TRUE` or `FALSE`, default is `FALSE`. 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.
`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 input giving the position for the new columns, can be one of c("first", "left", "right", "last", "keep"). The input can be abbreviated.
`overwrite`	`TRUE` or `FALSE`, default is `FALSE`. 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.
`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 gjeldende_prodnr8. has been added in the column to the right of the column with the 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	The year that should be selected as integer. Can be given as one year, the first and last year or a range of years.
hensikt	Vector with one or more hensiktkoder. Sub-hensikter are not included and must be explicitly mentioned.
db	The database for which the query is built. Currently only the value "PJS" is accepted.

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 should 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`	One year or a vector with years giving the first and last years that should be selected as integer.
`analytt`	One or more analyttkode given as a character. If sub-analytter should be included, end the code with %.
`hensikt`	Vector with specific hensikter. If sub-hensikter should be included, end the code with %. Can be `NULL`.
`metode`	Vector with specific metoder. Can be `NULL`.
`db`	The database for which the query is built. Currently only the value "PJS" is accepted.

Details

The function builds the SQL syntax to select all PJS-journals concerning one infection and/o 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-statement fom v2_sak_m_res and v_sakskonklusjon to 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_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`	One year or a vector with the first and last year that should be selected as integer vector.
`varname`	The PJS variable name of the variable in PJS from which the coded values should be selected.
`db`	The database for which the query is built. Currently only the value "PJS" is accepted.
`values`	The value of the codes that should be selected given as character. 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 variabel in PJS with hierarchical codes. When the code value including sub codes should be selected, add "%" to the 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` and `build_query_one_disease`. 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").
`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,
#   one with sak, prove, konklusjon and one with sak, prove, undersokelse and resultat
sak_prove_konkl <- 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)

sak_prove_und_res <- 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"
)

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

Arguments

`from_path`	Path for the produksjonstilskuddsregister.
`to_path`	Target path for the files with the produksjonstilskuddsregister.
`Pkode_year`	The year(s) from which the register should be read. Options is "last", or a vector with one or more years.
`Pkode_month`	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.

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.

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

Value

`read_Prodtilskudd` 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)
```

`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 "exclude" are samples from abroad be excluded. Allowed values are c("exclude", "include").
`quality`	If "exclude" are samples registered as quality assurance and ring trials 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)
```

`get_PAT` *Manage personal access token (PAT) for internet services*

Description

Save or remove the current user's PAT for internet services in the the user profile.

Usage

```
get_PAT(service)

remove_PAT(service)

set_PAT(service)
```

Arguments

`service`	Name of the internet service, for example "GitHub". For internet services where one don't use the premade wrappers, the name can be chosen freely, but must be the same as used in `get_PAT`

Details

For internet services like GitHub, personal access tokens can replace username and password when accessing the service. To simplify the access to the internet services when using R, the function `set_PAT` makes it possible to save the personal access token (PAT) in the user profile at the current machine. When the PAT has been saved in the user profile, the functions `get_PAT` will automatically get the PAT for use in code accessing the internet service.

The user profile is not copied between computers. Consequently, if a user runs scripts with `get_PAT` on different computers, the PAT has to be saved at each computer separately.

`set_PAT(service)` is used to set the PAT for a internet service. The PAT are input using windows and saved in the users profile at the current computer. When the PAT for the service has been changed, `set_PAT(service)` can be used to update the PAT.

`get_PAT(service)` is used to get the PAT for a internet service that previously has been saved in the users profile at the current computer.

`remove_PAT(service)` is used to delete the PAT for a internet service from the user's profile.

Value

`set_PAT` The PAT for a internet service are saved in the user profile at the current computer.

`get_PAT` The PAT for a internet service are fetched from the user profile at the current computer to be used in R-scripts.

`remove_PAT` The PAT for a internet service are deleted from the user profile at the current computer.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
set_PAT("GitHub")

get_PAT("GitHub")

remove_PAT("GitHub")

## End(Not run)
```

`login` *Log in to data base services*

Description

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

Usage

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

login_PJS()

login_EOS()

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

login_by_credentials_PJS()

login_by_credentials_EOS()

login_by_input(
  dbservice,
  dbdriver = NULL,
  db = NULL,
  dbserver = NULL,
  dbport = NULL,
  dbprotocol = NULL,
  dbtext = NULL
)

login_by_input_PJS()

login_by_input_EOS()
```

Arguments

`dbservice`	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`	Name of database engine
`db`	Name of database
`dbserver`	Name of database server
`dbport`	Port
`dbprotocol`	Protocol to be used
`dbtext`	used in login with input. Gives the possibility of showing another name than the dbservice in the windows asking for username and password.

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 general functions where all necessary specifications like server name and database name of the database must be input. The database provider can give information on what specifications that has to be used. This can be used to log in to many different databases. In the case that one login to a database service for which the connection parameters have been predefined (i.e. PJS, EOS, sea_sites and Fallvilt), it will be sufficient to provide the parameter dbservice.

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 general functions where all necessary specifications like server name and database name of the database must be input. In the case that one login to a database service for which the connection parameters have been predefined (i.e. PJS, EOS, sea_sites and Fallvilt), it will be sufficient to provide the parameter dbservice. The user is always asked to input username and password.

login_by_credentials is general functions where all necessary specifications like server name and database name of the database must be input. In the case that one login to a database service for which the connection parameters have been predefined (i.e. PJS, EOS, sea_sites and Fallvilt), 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 preset. The user only need to input username and password or if the username and password for journal_rapp/PJS are 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.

login_EOS, login_by_input_EOS, and login_by_credentials_EOS are wrappers for the functions above where the specifications for the database EOS have been preset. The user only need to input username and password or if the credentials are saved in the users profile by set_credentials_EOS, no input is needed.

The login functions returns an open ODBC-channel to the database service. The database can then be queried by using functions in the package RODBC.

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

Value

An open ODBC-channel to the database service.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

See Also

set_credentials

Examples

```
## Not run:
require(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.

`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 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/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 thathave 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 7 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

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_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

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_Prodtilskudd` 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)
```

`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()

set_credentials_EOS()
```

Arguments

`dbservice`	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.

`set_credentials_EOS` is a wrapper for `set_credentials(dbservice)` used to set the username and password for EOS. EOS has tables with surveillance data reported to the Norwegian Food Safety Authority.

`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)

set_credentials_PJS()

set_credentials_EOS()

remove_credentials("PJS")

## End(Not run)

# NULL
```

`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)
```

Arguments

`datasource`	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 = "?")`.

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.

Value

The full path for the directory at NVI's network. The path ends with "/".

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")

## 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_one_disease` or `build_query_hensikt`.

Usage

```
set_disease_parameters(
  hensikt2select = NULL,
  utbrudd2select = NULL,
  metode2select = NULL,
  analytt2select = NULL,
  art2select = NULL,
  file = NULL
)
```

Arguments

`hensikt2select`	Vector with specific hensikter. If sub-codes should be included, end the code with %. Can be `NULL`.
`utbrudd2select`	String with an utbrudd ID. Can be `NULL`.
`metode2select`	Vector with specific metoder. Can be `NULL`.
`analytt2select`	Vector with one or more analyttkode given as a character. If sub-codes should be included, end the code with %. Can be `NULL`.
`art2select`	Vector with one or more artkode given as a character. If sub-codes should be included, end the code with %. `NA` can be combined with another artkode. Can be `NULL`.
`file`	path and file name for an R script that can be sourced and that sets the parameters `hensikt2select`, `utbrudd2select`, `metode2select`, and `analytt2select`. Can be `NULL`.

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.

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 specific infectious agent(s) or disease. 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. If sub-codes of analytt or hensikt should be included, end the code with %.

The selection parameters can be input values for dedicated arguments. For input parameters `hensikt2select`, `utbrudd2select`, `metode2select`, and `analytt2select`, 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 at one disease at a time.

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 or if `property = "colclasses"` the path and filname of the csv-file used as data source
`dbsource`	database source of data. Set to data if not specifically specified. Needed if translation to column names is dependent on data source
`standards`	to input alternative standard tables to column_standards
`property`	Property of the column that should be standardized, currently c("colnames", "colclasses", "collabels", "colwidths_Excel", "colorder").
`language`	Language for labels. Valid input are c("no", "en")
`exclude`	Used in combination with `property = "colorder"`. `exclude = TRUE` excludes all columns with no predefinedcolorder.
`...`	Other arguments to be passed read.csv2 when `property = "colclasses"`.

Details

Experimental, the standardization table is under development. This version only works when being connected to the NVI network.

Variables in internal and external data sources uses different variable names for the same content. `Standarddize_columns` standardizes column names for use in scripts. It will be further developed to standardize column labels and column widths for both Excel and DT. Furthermore, input values for the parameter `colClasses =` for `read.csv2` 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 one must give an additional argument like `fileEncoding = "latin"`.

`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 changed automatiacally but can be changed by using a colname statement (see help). If no standard column label 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::.colwidth()`. 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 standardized. 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.

`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::.colwidth()`.

`property = "colorder"`. A data frame with column names in predefined order. If exclude = TRUEonly columns withh 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_PJSdata` *Standardizing PJS-data*

Description

Standardizing PJS-data. This standardizing should always be performed. Other functions used for further preparation of PJSdata, like `chose_PJS_levels`, and `exclude_form_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 with data extracted from PJS.
`dbsource`	If specified, this will be used for fetching standard column names by `standardize_columns`.

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)
```

Index