Package 'NVIpjsr'

August 19, 2024

Title Tools to facilitate working with PJS data

Version 0.1.1

Date 2024-08-19

Description Tools for retrieving, standardising, wrangling, preparing and reporting PJS data and EOS data. NVIpjsr was created by separating out PJS-functions from NVIdb. Further development of and creation of new PJS-functions will take place within NVIpjsr.

License BSD_3_clause + file LICENSE

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

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

Depends R (>= 4.1.0)

Encoding UTF-8

Language en-GB

Roxygen list(markdown = FALSE)

RoxygenNote 7.3.1

Imports checkmate (>= 2.1.0),
data.table,
DBI,
dplyr,
stats,
NVIcheckmate (>= 0.7.0),
NVIdb (>= 0.11.2)

Suggests covr,
desc,
devtools,
knitr,
R.rsp,
remotes,
rmarkdown,
testthat (>= 3.0.0),
tibble,
usethis,
NVIpackager (>= 0.5.0),
NVIrpackages (>= 0.4.0)

Remotes github::NorwegianVeterinaryInstitute/NVIcheckmate,
github::NorwegianVeterinaryInstitute/NVIdb,
github::NorwegianVeterinaryInstitute/NVIpackager,
github::NorwegianVeterinaryInstitute/NVIrpackages

VignetteBuilder knitr,
R.rsp

Config/testthat/edition 3

LazyData true

Contents

`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 = file.path(NVIdb::set_dir_NVI("Provedata_Rapportering", slash = FALSE),
    "FormaterteData"),
  to_path = NULL
)

read_PJS_codes_2_text(
  filename = "PJS_codes_2_text.csv",
 from_path = file.path(NVIdb::set_dir_NVI("Provedata_Rapportering", slash = FALSE),
    "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 files used to generate the translation table. Defaults to file.path(NVIdb::set_dir_NVI("Provedata_Rapportering", slash = FALSE), "FormaterteData").

to_path [character(1)]
Path to which the source files for generating the translation table should be copied.

... Other arguments to be passed to utils::read.csv2.

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)
```

`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. The select statements can thereafter be used to query journal_rapp/PJS using `DBI::dbGetQuery` when using odbc or `RODBC::sqlQuery` when using `RODBC`.

Value

A list with select statements for "v2_sak_m_res" and "v_sakskonklusjon", respectively.

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 select statements with SQL syntax to select all PJS-journals concerning one infection and/or disease from PJS. The select statements can thereafter be used to query journal_rapp/PJS using `DBI::dbGetQuery` when using `odbc` or `RODBC::sqlQuery` when using `RODBC`.

The select statements are build to select 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.

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 select statements with SQL syntax to select all PJS-saker regarding a disease outbreak from PJS. The select statements can thereafter be used to query journal_rapp/PJS using DBI::dbGetQuery when using odbc or RODBC::sqlQuery when using RODBC.

The select statements are build to select all journals within an outbreak where an 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.

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 `NVIdb::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(DBI)
library(NVIdb)

# Read from PJS
journal_rapp <- NVIdb::login_by_credentials("PJS", dbinterface = "odbc")
PJSdata <- DBI::dbGetQuery(
  con = journal_rapp,
  statement = paste("select *",
                    "from V2_SAK_M_RES",
                "where aar = 2020 and ansvarlig_seksjon = '01' and innsendelsesnummer = 1"))
DBI::dbDisconnect(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)
```

`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, eier_lokalitetnr 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_by_credentials_PJS` *Log in to PJS /journal_rapp*

Description

Log in to NVI's data base service: journal_rapp/PJS.

Usage

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

Arguments

`dbinterface`	[`character(1)`] The R-package that is used for interface towards the data base. Defaults to `NULL` which implies using `RODBC`.
`...`	Other arguments to be passed to `login_by_credentials`.

Details

`login_by_credentials_PJS` is wrapper for `NVIdb::login_by_credentials` and is the same as `NVIdb::login_by_credentials("PJS")`. It uses predefined connection parameters for the PJS database and it is therefore dependent on `NVIconfig` being installed. 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 by `NVIdb::set_credentials("PJS")`. If you would like to be asked for username and password, you should use `NVIdb::login("PJS")`.

`login_by_credentials_PJS` returns an open ODBC-channel to journal_rapp/PJS. 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` or `RODBC`. The default for journal_rapp/PJS is `RODBC` to obtain backward compatibility. However, it is recommended to use `odbc` which is faster.

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

Value

An open ODBC-channel to journal_rapp/PJS.

Author(s)

Petter Hopp Petter.Hopp@vetinst.no

Examples

```
## Not run:
library(odbc)
library(DBI)
journal_rapp <- login_by_credentials_PJS(dbinterface = "odbc")
# Reads hensiktregistret from PJS
hensikter <- DBI::dbGetQuery(con = journal_rapp,
                             statement = "select * from v_hensikt")
DBI::dbDisconnect(journal_rapp)

## End(Not run)
```

`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 `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 `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 `NVIpjsr`

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 `choose_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 `NVIpjsr`

`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(NVIdb::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

`retrieve_PJSdata` *Retrieves data from PJS*

Description

Retrieves and standardises PJS data. `retrieve_PJSdata` is a wrapper for several `NVIdb` - and `NVIpjsr` - 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, ...)
```

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`.
`...`	Other arguments to be passed to the underlying functions: `NVIdb::login("PJS")` and `exclude_from_PJSdata`.

Details

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

- Constructs the select statement by a build_query-function (see details) and selection parameters.
- Creates an open ODBC-channel using `NVIdb::login("PJS")`.
- Retrieves the data using the select statement constructed above.
- Standardises the data using `NVIdb::standardize_columns`.
- 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 in the user profile at the current computer using `NVIdb::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 "%" to indicate 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_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_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 NVIdb::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 NVIdb::standardize_columns.

Details

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

- The column names are standardised using NVIdb::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 to "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 `NVIdb::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 `NVIdb::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