Summarise and analyse clinical trial information

Ralf Herold

2023-06-24

General information on the `ctrdata` package is available here: https://github.com/rfhb/ctrdata.

Remember to respect the registers' terms and conditions (see `ctrOpenSearchPagesInBrowser(copyright = TRUE)`). Please cite this package in any publication as follows: Ralf Herold (2022). ctrdata: Retrieve and Analyze Clinical Trials in Public Registers. R package version 1.9.0. https://cran.r-project.org/package=ctrdata

Preparations

Here using MongoDB, which is faster than SQLite, can handle credentials, provides access to remote servers and can directly retrieve nested elements from paths. See README.md and Retrieve clinical trial information for examples using SQLite. Also PostgreSQL can be used as database, see Install R package ctrdata.

```
db <- nodbi::src_mongo(
  url = "mongodb://localhost",
  db = "my_database_name",
  collection = "my_collection_name"
)
db
# MongoDB 5.0.6 (uptime: 387411s)
# URL: mongodb://localhost
# Database: my_database_name
# Collection: my_collection_name
```

See Retrieve clinical trial information for more details.

```
library(ctrdata)

# These two queries are similar, for completed interventional (drug)
# trials with children with a neuroblastoma from either register
ctrLoadQueryIntoDb(
  # using queryterm and register ...
  queryterm = "query=neuroblastoma&age=under-18&status=completed",
  register = "EUCTR",
  euctrresults = TRUE,
  con = db
)
ctrLoadQueryIntoDb(
  # or using full URL of search results
  queryterm =
    "https://classic.clinicaltrials.gov/ct2/results?cond=neuroblastoma&recrs=e&age=0&intr=Drug",
```

```
  con = db
)
dbQueryHistory(con = db)
#       query-timestamp query-register query-records                                                query-term
# 1 2022-03-19 10:53:29          EUCTR           146 query=neuroblastoma&age=under-18&status=completed
# 2 2022-03-19 10:53:39          CTGOV           200        cond=neuroblastoma&recrs=e&age=0&intr=Drug
```

Find fields / variables of interest

Specify a part of the name of a variable of interest; all variables including deeply nested variable names are searched.

```
dbFindFields(namepart = "date", con = db)
# Finding fields in database collection (may take some time)
# Field names cached for this session.
#                                                                    EUCTR
#                                  "n_date_of_competent_authority_decision"
#                                                                    EUCTR
#                                   "p_date_of_the_global_end_of_the_trial"
#                                                                    EUCTR
#                                      "trialInformation.analysisStageDate"
#                                                                    EUCTR
#                                   "trialInformation.globalEndOfTrialDate"
#                                                                    EUCTR
#                                 "trialInformation.primaryCompletionDate"
#                                                                    EUCTR
#                                   "trialInformation.recruitmentStartDate"
#                                                                    EUCTR
# "x6_date_on_which_this_record_was_first_entered_in_the_eudract_database"
#                                                                    EUCTR
#                                      "n_date_of_ethics_committee_opinion"
#                                                                    CTGOV
#                                                        "completion_date"
#                                                                    CTGOV
#                                                   "last_update_posted"
#                                                                    CTGOV
#                                                 "last_update_submitted"
#                                                                    CTGOV
#                                               "last_update_submitted_qc"
#                                                                    CTGOV
#                                                "primary_completion_date"
#                                                                    CTGOV
#                                           "required_header.download_date"
#                                                                    CTGOV
#                                                             "start_date"
#                                                                    CTGOV
#                                                       "verification_date"
#                                                                    CTGOV
#                 "provided_document_section.provided_document.document_date"
```

The search for fields is cached and thus accelerated during the R session, as long as no new `ctrLoadQueryIntoDb()` is executed.

Data frame from database

The fields of interest can be obtained from the database and are represented in an R data.frame:

```
result <- dbGetFieldsIntoDf(
  c(
    "f41_in_the_member_state",
    "f422_in_the_whole_clinical_trial",
    "a1_member_state_concerned",
    "p_end_of_trial_status",
    "n_date_of_competent_authority_decision",
    "a2_eudract_number",
    "overall_status",
    "start_date",
    "primary_completion_date"
  ),
  con = db
)
```

Metadata from data frame

The objects returned by functions of this package include attributes with metadata to indicate from which database, table / collection and query details. Metadata can be reused in R.

```
attributes(result)
# [...]
# $`ctrdata-dbname`
# [1] "my_database_name"

# $`ctrdata-table`
# [1] "my_collection_name"

# $`ctrdata-dbqueryhistory`
#       query-timestamp query-register query-records                                          query-term
# 1 2022-03-19 10:53:29          EUCTR           146 query=neuroblastoma&age=under-18&status=completed
# 2 2022-03-19 10:53:39          CTGOV           200        cond=neuroblastoma&recrs=e&age=0&intr=Drug
```

In the database, the variable "_id" is the unique index for a record. This "_id" is the NCT number for CTGOV records (e.g., "NCT00002560"), and it is the EudraCT number for EUCTR records including the postfix identifying the EU Member State (e.g., "2008-001436-12-NL").

It is relevant to de-duplicate records because a trial can be registered in both CTGOV and EUCTR, and can have records by involved country in EUCTR.

De-duplication is done at the analysis stage because this enables to select if a trial record should be taken from one or the other register, and from one or the other EU Member State.

The basis of de-duplication is the recording of additional trial identifiers in supplementary fields (variables), which are checked and reported when using function `dbFindIdsUniqueTrials()`:

```
# Obtain de-duplicated trial record ids
ids <- dbFindIdsUniqueTrials(
  preferregister = "EUCTR",
  con = db
```

```
)
# Searching for duplicate trials...
#  - Getting trial ids (may take some time), 346 found in collection
#  - Finding duplicates among registers' and sponsor ids...
#  - 106 EUCTR _id were not preferred EU Member State record for 42 trials
#  - Keeping 187 / 40 records from CTGOV / EUCTR
# = Returning keys (_id) of 227 records in collection "my_collection_name"
```

The unique ids can be used like this to de-duplicate the data.frame created above:

```
# Eliminate duplicate trials records:
result <- result[result[["_id"]] %in% ids, ]
#
nrow(result)
# [1] 227
```

Simple analysis - dates

In a data.frame generated with `dbGetFieldsIntoDf()`, fields are typed as dates, logical, character or numbers.

```
str(result)
# 'data.frame':   227 obs. of  10 variables:
# $ _id                                  : chr  "2004-004386-15-DE" "2005-000915-80-IT" "2005-001267-6
# $ f41_in_the_member_state              : int  NA NA 5 37 70 24 100 35 10 24 ...
# $ f422_in_the_whole_clinical_trial     : int  230 NA 12 67 70 NA 100 156 2230 NA ...
# $ a1_member_state_concerned            : chr  "Germany - BfArM" "Italy - Italian Medicines Agency" .
# $ p_end_of_trial_status                : chr  "Completed" "Completed" "Completed" "Completed" ...
# $ n_date_of_competent_authority_decision: Date, format: "2005-07-08" "2005-04-21" "2005-07-08" "2006-
# $ a2_eudract_number                    : chr  "2004-004386-15" "2005-000915-80" "2005-001267-63" "20
# $ overall_status                       : chr  NA NA NA NA ...
# $ start_date                           : Date, format: NA NA NA NA ...
# $ primary_completion_date              : Date, format: NA NA NA NA ...
```

This typing facilitates using the respective type of data for analysis, for example of dates with base R graphics:

```
# Open file for saving
png("vignettes/nb1.png")
# Visualise trial start date
hist(
  result[["n_date_of_competent_authority_decision"]],
  breaks = "years"
)
box()
dev.off()
```

Merge corresponding fields from registers

The field "n_date_of_competent_authority_decision" used above exists only in EUCTR, yet it corresponds to the field "start_date" in CTGOV. Thus, to give the start of the trial, the two fields can be merged for analysis, using the convenience function `dfMergeTwoVariablesRelevel()` in `ctrdata` package:

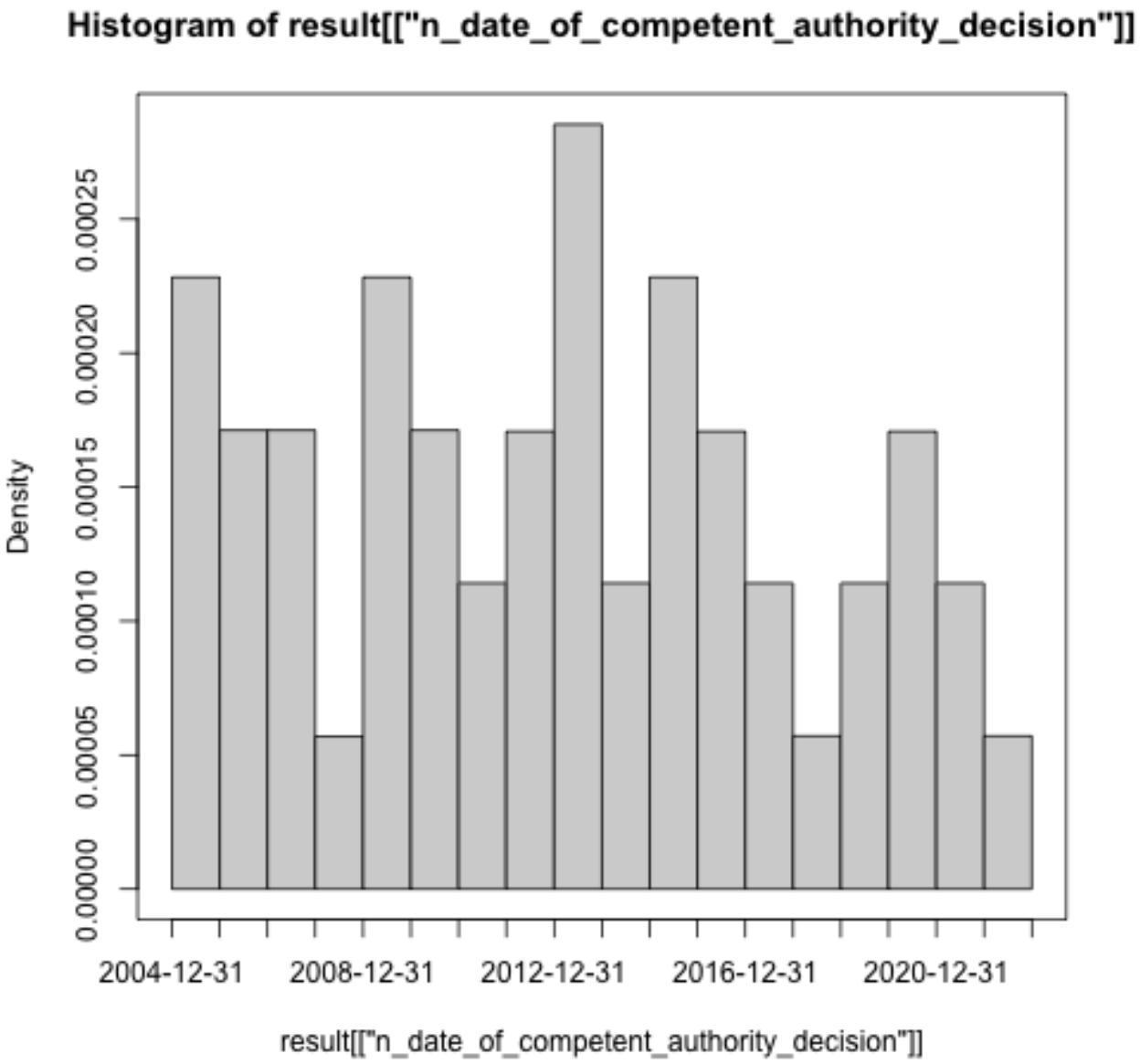

Figure 1: Histogram1

```r
# Merge two variables into a new variable:
result$trialstart <- dfMergeTwoVariablesRelevel(
  result,
  colnames = c(
    "n_date_of_competent_authority_decision",
    "start_date"
  )
)
# Unique values returned (first three): 2005-07-08, 2005-04-21, 2006-05-26

# Plot from both registers
png("vignettes/nb2.png")
hist(
  result[["trialstart"]],
  breaks = "years"
)
box()
dev.off()
```

In a more sophisticated use of `dfMergeTwoVariablesRelevel()`, values of the original variables can be mapped into new values of the merged variable, as follows:

```r
# First, define how values of the new, merged variable
# (e.g., "ongoing") will result from values of the
# original variable (e.g, "Recruiting):
mapped_values <- list(
  "ongoing" = c(
    "Recruiting", "Active", "Ongoing",
    "Active, not recruiting",
    "Enrolling by invitation", "Restarted"
  ),
  "completed" = c("Completed", "Prematurely Ended", "Terminated"),
  "other" = c(
    "Withdrawn", "Suspended", "No longer available",
    "Not yet recruiting", "Temporarily Halted",
    "Unknown status", "GB - no longer in EU/EEA"
  )
)

# Secondly, use the list of mapped
# values when merging two variable:
tmp <- dfMergeTwoVariablesRelevel(
  result,
  colnames = c(
    "overall_status",
    "p_end_of_trial_status"
  ),
  levelslist = mapped_values
)
# Unique values returned: completed, ongoing, other

table(tmp)
#   completed   ongoing     other
#         223         3         1
```

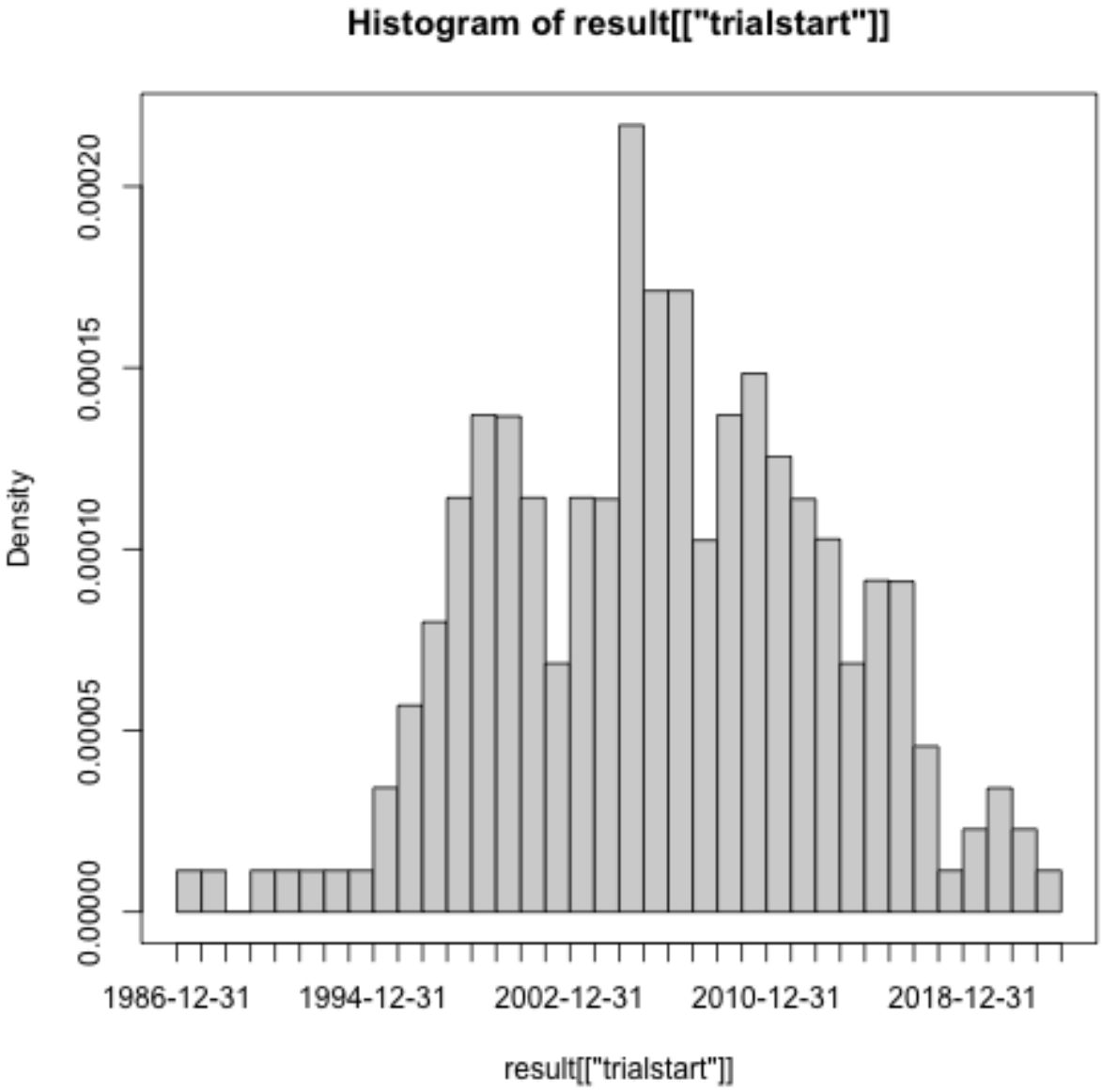

Figure 2: Histogram2

Annotations made by user

When using `ctrLoadQueryIntoDb()`, `ctrdata` adds to each record the fields `annotation` and `record_last_import`. The annotation field is a single string that the user specifies when retrieving trials (Retrieve clinical trial information). The user can specify to append, prefix or replace any existing annotations when a trial record is loaded again, see example below. The last date and time when the trial record was imported is updated automatically when using `ctrLoadQueryIntoDb()`. These fields can also be used for analysis. For example, string functions can be used for annotations, e.g. to split it into components. Since no annotations were specified when retrieving the trials in the steps above, there are so far no annotation fields and `stopifnodata` is set to `FALSE` to avoid the function raises an error to alert users:

```
ctrLoadQueryIntoDb(
  queryterm = "query=neuroblastoma&resultsstatus=trials-with-results",
  register = "EUCTR",
  euctrresults = TRUE,
  annotation.text = "test annotation",
  annotation.mode = "append",
  con = db
)

result <- dbGetFieldsIntoDf(
  fields = c(
    "annotation",
    "record_last_import"
  ),
  con = db
)

str(result)
# 'data.frame':   373 obs. of  3 variables:
#  $ _id                : chr  "2004-004386-15-DE" "2004-004386-15-ES" "2004-004386-15-GB" ...
#  $ annotation         : chr  "test annotation" "test annotation" "test annotation" ...
#  $ record_last_import: Date, format: "2022-03-20" "2022-03-20" "2022-03-20" "2022-03-20"
```

Analysing nested fields such as trial results

The registers represent clinical trial information by nesting fields (e.g., several reporting groups within several measures within one of several endpoints). A visualisation of this hierarchical representation for CTGOV is this:

```
dbFindFields("outcome", db)
# remotes::install_github("https://github.com/hrbrmstr/jsonview")
result <- dbGetFieldsIntoDf("clinical_results.outcome_list.outcome", db)
jsonview::json_tree_view(result[["clinical_results.outcome_list.outcome"]][
  result[["_id"]] == "NCT00520936"
])
```

The analysis of nested information such as the highlighted duration of response is facilitated with `ctrdata` as follows. The main steps are:

- transform nested information to a long, name-value data frame and then

Figure 3: CtgovNested

- identify where the measures of interest (e.g. duration of response, blue circles above) are located in the information hierarchy by specifying the name and value of fields (`wherename`, `wherevalue`) and finally
- obtain the value by specifying the name(s) of its value field(s) (`valuename`, red circles in figure above).

```
# 1. Create data frame from results fields.
# These are the key results fields from
# CTGOV and from EUCTR:
result <- dbGetFieldsIntoDf(
  fields = c(
    # CTGOV
    "clinical_results.baseline.analyzed_list.analyzed.count_list.count",
    "clinical_results.baseline.group_list.group",
    "clinical_results.baseline.analyzed_list.analyzed.units",
    "clinical_results.outcome_list.outcome",
    "study_design_info.allocation",
    # EUCTR
    "@attributes.eudractNumber",
    "trialInformation.populationAgeGroup",
    "subjectDisposition.recruitmentDetails",
    "baselineCharacteristics.baselineReportingGroups.baselineReportingGroup",
    "endPoints.endPoint",
    "trialChanges.hasGlobalInterruptions",
    "subjectAnalysisSets",
    "adverseEvents.seriousAdverseEvents.seriousAdverseEvent"
  ),
  con = db
)
# Keep only unique trial records
result <- result[result[["_id"]] %in% dbFindIdsUniqueTrials(con = db), ]
# [...]
#  - Keeping 184 / 50 records from CTGOV / EUCTR
# = Returning keys (_id) of 234 records in collection "my_collection_name"

# 2. The columns of the results data frame
# contain nested lists of fields, example:
str(result[["endPoints.endPoint"]][1])

# All nested data are transformed to a long,
# name-value data.frame (resulting in several
# hundred rows per trial record):
long_result <- dfTrials2Long(df = result)
# Total 99006 rows, 143 unique names of variables

# 3. Obtain values for fields of interest where
# they related to measures of interest. The
# parameters can be regular expressions.
dor <- dfName2Value(
  df = long_result,
  wherename = paste0(
    "clinical_results.outcome_list.outcome.measure.title|",
    "endPoints.endPoint.title"
  ),
  wherevalue = "duration of response",
```

```
  valuename = paste0(
    "clinical_results.*category_list.category.measurement_list.measurement.value|",
    "endPoints.*armReportingGroup.tendencyValues.tendencyValue.value"
  )
)

# Duration of response has been reported variably in
# months and days. Here, just select trials reporting
# duration of response in months:
dor <- dor[
  grepl("months",
    dfName2Value(
      df = long_result,
      wherename = paste0(
        "clinical_results.*outcome.measure.title|",
        "endPoints.endPoint.title"
      ),
      wherevalue = "duration of response",
      valuename = paste0(
        "clinical_results.*measure.units|",
        "endPoints.endPoint.unit"
      )
    )[["value"]],
    ignore.case = TRUE
  ),
]

# the identifier is generated to identify repeated and
# nested elements in a trial, so that it permits to
# find the valuename that corresponds to the item
# for which wherename has wherevalue)
dor[, c("_id", "identifier", "value")]
#                 _id identifier value
# 1 2010-019348-37-IT        8.2   7.0
# 2 2010-019348-37-IT        8.3   8.1
# 5 2013-003595-12-SK        6.2 999.0
# 6 2013-003595-12-SK        6.3 999.0
# 7 2013-003595-12-SK        6.4 999.0
# 8 2013-003595-12-SK        6.5 999.0
# 9 2013-003595-12-SK        6.6 999.0
```

Analysing primary endpoints

Text analysis has to be used for many fields of trial information from the registers. Here is an example to simply categorise the type of primary endpoint. In addition, the number of subjects are extracted and compared by type of primary endpoint.

```
# Several "measure" entries are in "primary_outcome".
# They are concatenated into a list when specifying
# the JSON path "primary_outcome.measure"
result <- dbGetFieldsIntoDf(
  c(
    # CTGOV
```

```
    "primary_outcome.measure",
    "enrollment",
    # EUCTR
    "e51_primary_end_points",
    # "f11_trial_has_subjects_under_18"
    "f11_number_of_subjects_for_this_age_range"
  ),
  con = db
)

# De-duplicate
result <- result[
  result[["_id"]] %in%
    dbFindIdsUniqueTrials(con = db),
]

# Merge primary endpoint (pep)
result$pep <- dfMergeTwoVariablesRelevel(
  df = result,
  colnames =
    c(
      "primary_outcome.measure",
      "e51_primary_end_points"
    )
)

# Merge number of subjects
result$nsubj <- dfMergeTwoVariablesRelevel(
  df = result,
  colnames =
    c(
      "enrollment",
      "f11_number_of_subjects_for_this_age_range"
    )
)

# For primary endpoint of interest,
# use regular expression on text:
result$pep_is_efs <- grepl(
  pattern = "((progression|event|relapse|recurrence|disease)[- ]free)|pfs|dfs|efs)",
  x = result$pep,
  ignore.case = TRUE
)

# Tabulate
table(result$pep_is_efs)
# FALSE  TRUE
#   184    21

# Plot
library(ggplot2)
ggplot(
  data = result,
```

```
  aes(
    x = nsubj,
    y = pep_is_efs
  )
) +
  geom_boxplot() +
  scale_x_log10()
ggsave("vignettes/boxpep.png", width = 6, height = 4)
```

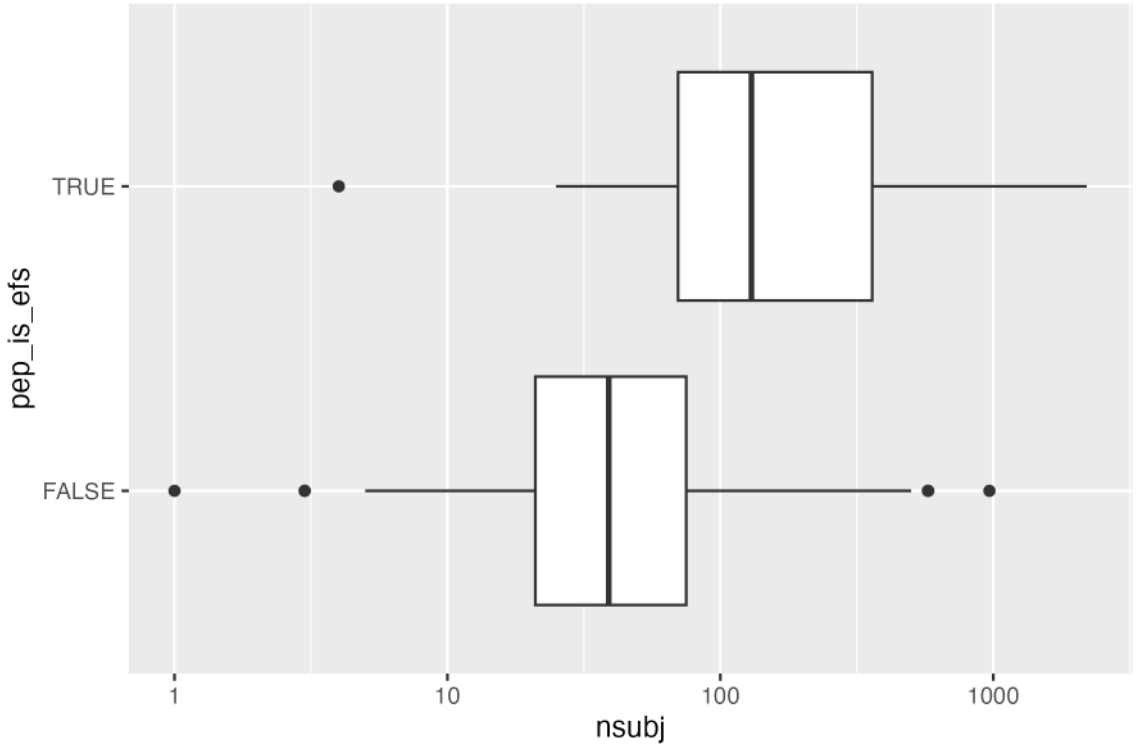

Figure 4: BoxPEP

Investigational or authorised medicinal product?

The information about the status of authorisation (licensing) of a medicine used in a trial is recorded in EUCTR in the field `dimp.d21_imp_to_be_used_in_the_trial_has_a_marketing_authorisation`. A corresponding field in CTGOV is not known. The status is in the tree starting from the `dimp` element.

```
# Get results
result <- dbGetFieldsIntoDf(
  fields =
    c(
      "a1_member_state_concerned",
      "n_date_of_competent_authority_decision",
      "dimp.d21_imp_to_be_used_in_the_trial_has_a_marketing_authorisation",
      "x6_date_on_which_this_record_was_first_entered_in_the_eudract_database",
      "f422_in_the_whole_clinical_trial",
```

```
      "a2_eudract_number"
    ),
  con = db
)

# Find first date of authorisation in EU member state
tmp <- aggregate(
  result[["n_date_of_competent_authority_decision"]],
  by = list(result[["a2_eudract_number"]]),
  FUN = function(x) min(x)
)
result <- merge(
  x = result,
  y = tmp,
  by.x = "a2_eudract_number",
  by.y = "Group.1",
  all.x = TRUE
)
result[["startdatefirst"]] <- dfMergeTwoVariablesRelevel(
  df = result,
  colnames = c(
    "x",
    "x6_date_on_which_this_record_was_first_entered_in_the_eudract_database"
  )
)

# Now de-duplicate
result <- result[
  result[["_id"]] %in%
    dbFindIdsUniqueTrials(
      include3rdcountrytrials = FALSE,
      con = db
    ),
]

# How many of the investigational medicinal product(s)
# used in the trial are authorised?
number_authorised <- sapply(
  result[["dimp.d21_imp_to_be_used_in_the_trial_has_a_marketing_authorisation"]],
  function(i) length(i[i])
)
table(number_authorised, exclude = "")
# number_authorised
#  0  1  2  3  4  6  8 15
# 20 15  4  3  1  1  1  1
result[["any_authorised"]] <- number_authorised > 0

# Plot
library(ggplot2)
library(scales)
ggplot(
  data = result,
  aes(
```

```
    x = startdatefirst,
    fill = any_authorised
  )
) +
  scale_x_date(
    breaks = breaks_width(width = "2 years"),
    labels = date_format("%Y")
  ) +
  geom_histogram(binwidth = 2 * 365.25) +
  labs(
    title = "Neuroblastoma trials in EU",
    x = "Year of trial authorisation (or entered in EUCTR)",
    y = "Number of trials",
    fill = "Medicine\nauthorised?"
  )
ggsave("vignettes/nbtrials.png", width = 6, height = 4)
```

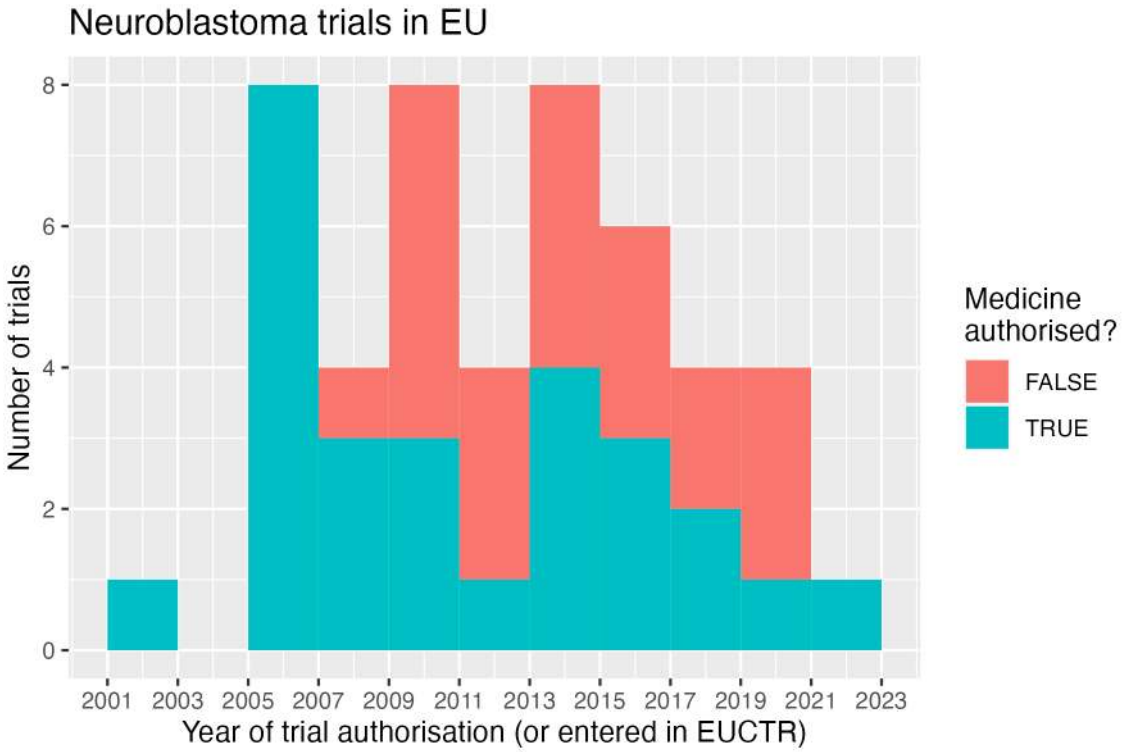

Figure 5: HistogramNBtrials

Analyses using aggregation pipeline and mapreduce

Here is an example of analysis functions that can be run directly on a MongoDB server, which are fast and do not consume R resources.

```
# Load library for database access
library(mongolite)
```

```
# Create R object m to access the
# collection db created above:
m <- mongo(
  url = paste0(db[["url"]], "/", db[["db"]]),
  collection = db[["collection"]]
)

# Number of records in  collection:
m$count()
# [1] 404

# Number of EUCTR records, using JSON for query:
m$count(query = '{"_id": {"$regex": "[0-9]{4}-[0-9]{6}-[0-9]{2}",
                 "$options": "i"}}')
# [1] 196

# Alternative:
m$count(query = '{"ctrname": "EUCTR"}')
# [1] 196

# Number of CTGOV records:
m$count(query = '{"_id": {"$regex": "NCT[0-9]{8}",
                 "$options": "i"}}')
# [1] 207

# Alternative:
m$count(query = '{"ctrname": "CTGOV"}')
# [1] 207

# To best define regular expressions for analyses,
# inspect the field (here, primary_outcome.measure):
# Regular expressions ("$regex") are case insensitive ("i")
head(
  m$distinct(
    key = "primary_outcome.measure",
    query = '{"ctrname": "CTGOV"}'
  )
)
# [1] "- To demonstrate that 123I-mIBG planar scintigraphy is sensitive and ..."
# [2] "1-year Progression-free Survival"
# [3] "Ability of iodine I 131 metaiodobenzylguanidine to provide palliative therapy"
# [4] "Acute and late toxicities"
# [5] "Adverse events as a measure of safety/tolerability"
# [6] "Adverse events as assessed by (CTCAE) version 4.0"
```

Aggregation

The following example uses the aggregation pipeline in MongoDB. See here for details on mongo's aggregation pipeline: https://docs.mongodb.org/manual/core/aggregation-pipeline/

```
# Total count of PFS, EFS, RFS or DFS
out <- m$aggregate(
```

```
  # Count number of documents in collection that
  # matches in primary_outcome.measure the
  # regular expression,
  pipeline =
    '[{"$match": {"primary_outcome.measure":
      {"$regex": "(progression|event|relapse|recurrence|disease)[- ]free",
                 "$options": "i"}}},
      {"$group": {"_id": "null", "count": {"$sum": 1}}}]'
)
out
#    _id count
# 1 null    17

# List records of trials with overall survival
# as primary endpoint, and list start date
out <- m$aggregate(
  pipeline =
    '[{"$match": {"primary_outcome.measure":
      {"$regex": "overall survival", "$options": "i"}}},
      {"$project": {"_id": 1, "start_date": 1}}]'
)
head(out)
#           _id      start_date
# 1 NCT00445965    January 2006
# 2 NCT00499616 October 8, 2007
# 3 NCT00793845     August 2008
# 4 NCT00923351    June 2, 2007
```

Mapreduce

The following examples uses the map-reduce operation in MongoDB. Note this is deprecated starting in MongoDB 5. More information: https://docs.mongodb.com/manual/core/map-reduce/ Thus, below is the same query expressed as aggregation pipeline.

```
# Count number of trials by number of study
# participants in bins of hundreds of participants:
hist <- m$mapreduce(
  map = "function() {emit(Math.floor(this.enrollment/100)*100, 1)}",
  reduce = "function(id, counts) {return Array.sum(counts)}"
)
# read as: 135 trials had 0 to less than 100 participants
hist[order(hist[["_id"]]), ]
#   _id value
# 8   0   135
# 1 100    28
# 5 200     5
# 2 300     1
# 9 400     4
# 6 500     4
# 7 600     1
# 3 900     1
# 4 NaN   225
```

```
# same but as aggregation pipeline,
# future proof with MongoDB
m$aggregate(pipeline = '
[
  {
    "$project": {
      "flooredNumber": {
        "$multiply": [
          {
            "$floor": {
              "$divide": [
                {
                  "$toInt": "$enrollment"
                },
                100
              ]
            }
          },
          100
        ]
      }
    }
  },
  {
    "$group": {
      "_id": "$flooredNumber",
      "count": {
        "$count": {}
      }
    }
  },
  {
    "$sort": {
      "_id": 1
    }
  }
]
')
#   _id count
# 1  NA   225
# 2   0   135
# 3 100    28
# 4 200     5
# 5 300     1
# 6 400     4
# 7 500     4
# 8 600     1
# 9 900     1
```