Summarise and analyse clinical trial information

Ralf Herold

2025-02-15

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 (2025). ctrdata: Retrieve and Analyze Clinical Trials in Public Registers. R package version 1.20.0. https://cran.r-project.org/package=ctrdata

Preparations

Here using MongoDB, which may be 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.

```r
db <- nodbi::src_mongo(
  url = "mongodb://localhost",
  db = "my_database_name",
  collection = "my_collection_name"
)
db
# MongoDB  8.0.4 (uptime: 298113s)
# URL: mongodb://localhost
# Database: my_database_name
# Collection: my_collection_name

# empty collection if exists
nodbi::docdb_delete(db, db$collection)
```

See Retrieve clinical trial information for more details.

```r
#
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
)
# [...]
# $n
# [1] 262

ctrLoadQueryIntoDb(
  # ...or using a full URL of search results
  queryterm = "https://clinicaltrials.gov/search?cond=neuroblastoma&intr=Drug&aggFilters=ages:child,sta
  con = db
)
# * Appears specific for CTGOV REST API 2.0
# * Found search query from CTGOV2: cond=neuroblastoma&intr=Drug&aggFilters=ages:child,status:com
# * Checking trials using CTGOV REST API 2.0, found 235 trials
# (1/3) Downloading in 1 batch(es) (max. 1000 trials each; estimate: 24 Mb total)
# Download status: 1 done; 0 in progress. Total size: 10.40 Mb (598%)... done!
# (2/3) Converting to NDJSON...
# (3/3) Importing records into database...
# JSON file #: 1 / 1
# = Imported or updated 235 trial(s)
# Updated history ("meta-info" in "my_collection_name")
# $n
# [1] 235

dbQueryHistory(con = db)
#       query-timestamp query-register query-records
# 1 2025-02-15 17:47:52          EUCTR           262             query=neuroblastoma&age=under-18&statu
# 2 2025-02-15 17:48:02         CTGOV2           235 cond=neuroblastoma&intr=Drug&aggFilters=ages:child
```

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. Set `sample = TRUE` (default) to rapidly execute the function in large databases.

```
#
dbFindFields(namepart = "date", sample = FALSE, con = db)
# Finding fields in database collection (may take some time) . . . . .
# Field names cached for this session.
#                                                                             EUCTR
#    "e231_full_title_date_and_version_of_each_substudy_and_their_related_objectives"
#                                                                             EUCTR
# "e231_full_title_date_and_version_of_each_substudy_and_their_related_objectives_es"
#                                                                             EUCTR
# "e231_full_title_date_and_version_of_each_substudy_and_their_related_objectives_it"
#                                                                             EUCTR
#                                            "n_date_of_competent_authority_decision"
#                                                                             EUCTR
#                                                "n_date_of_ethics_committee_opinion"
#                                                                             EUCTR
#                                             "p_date_of_the_global_end_of_the_trial"
#                                                                             EUCTR
#                                "trialChanges.globalAmendments.globalAmendment.date"
#                                                                             EUCTR
#                          "trialChanges.globalInterruptions.globalInterruption.date"
#                                                                               EUCTR
#                   "trialChanges.globalInterruptions.globalInterruption.restartDate"
#                                                                               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"
#                                                                              CTGOV2
#         "annotationSection.annotationModule.unpostedAnnotation.unpostedEvents.date"
#                                                                              CTGOV2
#  "derivedSection.miscInfoModule.submissionTracking.estimatedResultsFirstSubmitDate"
#                                                                              CTGOV2
#      "derivedSection.miscInfoModule.submissionTracking.firstMcpInfo.postDateStruct"
#                                                                              CTGOV2
# "derivedSection.miscInfoModule.submissionTracking.firstMcpInfo.postDateStruct.date"
#                                                                              CTGOV2
# "derivedSection.miscInfoModule.submissionTracking.firstMcpInfo.postDateStruct.type"
#                                                                              CTGOV2
#      "derivedSection.miscInfoModule.submissionTracking.submissionInfos.releaseDate"
#                                                                              CTGOV2
#        "derivedSection.miscInfoModule.submissionTracking.submissionInfos.resetDate"
#                                                                              CTGOV2
#                                "documentSection.largeDocumentModule.largeDocs.date"
#                                                                              CTGOV2
#                          "documentSection.largeDocumentModule.largeDocs.uploadDate"
#                                                                              CTGOV2
#                                 "protocolSection.statusModule.completionDateStruct"
#                                                                              CTGOV2
#                            "protocolSection.statusModule.completionDateStruct.date"
#                                                                              CTGOV2
#                            "protocolSection.statusModule.completionDateStruct.type"
#                                                                              CTGOV2
#                              "protocolSection.statusModule.dispFirstPostDateStruct"
#                                                                              CTGOV2
#                         "protocolSection.statusModule.dispFirstPostDateStruct.date"
#                                                                              CTGOV2
#                         "protocolSection.statusModule.dispFirstPostDateStruct.type"
#                                                                              CTGOV2
#                                  "protocolSection.statusModule.dispFirstSubmitDate"
#                                                                              CTGOV2
#                                "protocolSection.statusModule.dispFirstSubmitQcDate"
#                                                                              CTGOV2
#                             "protocolSection.statusModule.lastUpdatePostDateStruct"
#                                                                              CTGOV2
#                        "protocolSection.statusModule.lastUpdatePostDateStruct.date"
#                                                                              CTGOV2
#                        "protocolSection.statusModule.lastUpdatePostDateStruct.type"
#                                                                               CTGOV2
#                                  "protocolSection.statusModule.lastUpdateSubmitDate"
#                                                                               CTGOV2
#                           "protocolSection.statusModule.primaryCompletionDateStruct"
#                                                                               CTGOV2
#                      "protocolSection.statusModule.primaryCompletionDateStruct.date"
#                                                                               CTGOV2
#                      "protocolSection.statusModule.primaryCompletionDateStruct.type"
#                                                                               CTGOV2
#                            "protocolSection.statusModule.resultsFirstPostDateStruct"
#                                                                               CTGOV2
#                       "protocolSection.statusModule.resultsFirstPostDateStruct.date"
#                                                                               CTGOV2
#                       "protocolSection.statusModule.resultsFirstPostDateStruct.type"
#                                                                               CTGOV2
#                                "protocolSection.statusModule.resultsFirstSubmitDate"
#                                                                               CTGOV2
#                              "protocolSection.statusModule.resultsFirstSubmitQcDate"
#                                                                               CTGOV2
#                                       "protocolSection.statusModule.startDateStruct"
#                                                                               CTGOV2
#                                  "protocolSection.statusModule.startDateStruct.date"
#                                                                               CTGOV2
#                                  "protocolSection.statusModule.startDateStruct.type"
#                                                                               CTGOV2
#                                    "protocolSection.statusModule.statusVerifiedDate"
#                                                                               CTGOV2
#                              "protocolSection.statusModule.studyFirstPostDateStruct"
#                                                                               CTGOV2
#                         "protocolSection.statusModule.studyFirstPostDateStruct.date"
#                                                                               CTGOV2
#                         "protocolSection.statusModule.studyFirstPostDateStruct.type"
#                                                                               CTGOV2
#                                  "protocolSection.statusModule.studyFirstSubmitDate"
#                                                                               CTGOV2
#                                "protocolSection.statusModule.studyFirstSubmitQcDate"
```

The search for fields is cached and thus accelerated during the R session; calling `ctrLoadQueryIntoDb()` or changing `sample = ...` invalidates the cache.

Data frame from database

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

```
#
result <- dbGetFieldsIntoDf(
  fields = c(
    # EUCTR protocol-related information
    "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",
    #
    # EUCTR results-related information
    "trialInformation.recruitmentStartDate",
    "trialInformation.globalEndOfTrialDate",
    #
    # CTGOV2
    "protocolSection.statusModule.overallStatus",
    "protocolSection.statusModule.startDateStruct.date",
    "trialInformation.recruitmentStartDate",
    "protocolSection.statusModule.primaryCompletionDateStruct.date"
  ),
  con = db
)
# Querying database (11 fields)...
```

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)
# [...]
#
# $class
# [1] "data.frame"
#
# $`ctrdata-dbname`
# [1] "my_database_name"
#
# $`ctrdata-table`  <-- this attribute will be retired by end 2024
# [1] "my_collection_name"
#
# $`ctrdata-table-note`
# [1] "^^^ attr ctrdata-table will be removed by end 2024"
#
# $`ctrdata-collection`
# [1] "my_collection_name"
#
# $`ctrdata-dbqueryhistory`
#       query-timestamp query-register query-records
# 1 2025-02-15 17:47:52          EUCTR           262             query=neuroblastoma&age=under-18&statu
# 2 2025-02-15 17:48:02         CTGOV2           235 cond=neuroblastoma&intr=Drug&aggFilters=ages:child
```

De-duplicate records

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 all trial identifiers (may take some time), 497 found in collection
# - Finding duplicates among registers' and sponsor ids...
# - 200 EUCTR _id were not preferred EU Member State record for 64 trials
# - Keeping 62 / 0 / 0 / 0 / 216 records from EUCTR / CTGOV / ISRCTN / CTIS / CTGOV2
# = Returning keys (_id) of 278 records in collection "my_collection_name"

# Eliminate duplicate trials records:
result <- result[result[["_id"]] %in% ids, ]

nrow(result)
# [1] 278
```

An alternative is to calculate a column `isUniqueTrial` already at the time that a data.frame is created:

```
# Obtain data of interest
result <- dbGetFieldsIntoDf(
  # fields = c("<see above for example>", ...),
  calculate = ".isUniqueTrial",
  con = db
)
# Querying database (23 fields)...
# - Finding duplicates among registers' and sponsor ids...
# - 151 EUCTR _id were not preferred EU Member State record for 58 trials
# - Keeping 235 / 46 / 0 / 0 / 0 records from CTGOV2 / EUCTR / CTGOV / ISRCTN / CTIS

# Eliminate duplicate trials records:
result <- result[result[[".isUniqueTrial"]], ]

nrow(result)
# [1] 307
```

Reviewing a specific trial

It will often be useful to inspect all data for a single, for example to understand the meaning and relation of fields, and to see neighboring elements to fields of interest.

```
#
# Adding some trials from CTIS for a search similar to above
ctrLoadQueryIntoDb(
  queryterm = 'searchCriteria={"containAny":"neuroblastoma"}',
  register = "CTIS",
  con = db
)
# $n
# [1] 26
#
# $success
#  [1] "2024-519089-32-02" "2024-514917-36-00" "2024-518931-12-00" "2023-504246-64-02" "2024-517295-37-
#  [6] "2024-513470-22-00" "2024-513843-10-00" "2022-501694-39-01" "2024-511404-17-00" "2024-511975-14-
# [11] "2024-512102-25-00" "2024-515552-21-00" "2024-511259-16-00" "2023-507418-28-00" "2023-509673-22-
# [16] "2024-513141-37-00" "2024-511071-16-00" "2024-512095-35-00" "2023-507178-41-00" "2023-506778-11-
# [21] "2022-501725-21-00" "2023-504880-18-00" "2023-508587-29-00" "2022-502668-20-00" "2023-504246-64-
# [26] "2023-503684-42-00"
#
# $failed
# NULL
```

Identify a trial of interest by its `_id`, and use this for showing an interactive widget in the user's browser. This new functionality in `ctrdata` helps users to search and find any field and any value of the field in the trial. All data retrieved by `ctrdata` for the trial is shown. When a user selects fields and clicks on "Copy names…", the user can paste this directly into a call to function `dgGetFieldsIntoDf(...)`.

```
# Opens a web browser for user interaction.
# If the trial is not found in the database,
# it will be loaded from the register.
ctrShowOneTrial("2022-501725-21-00", con = db)
```

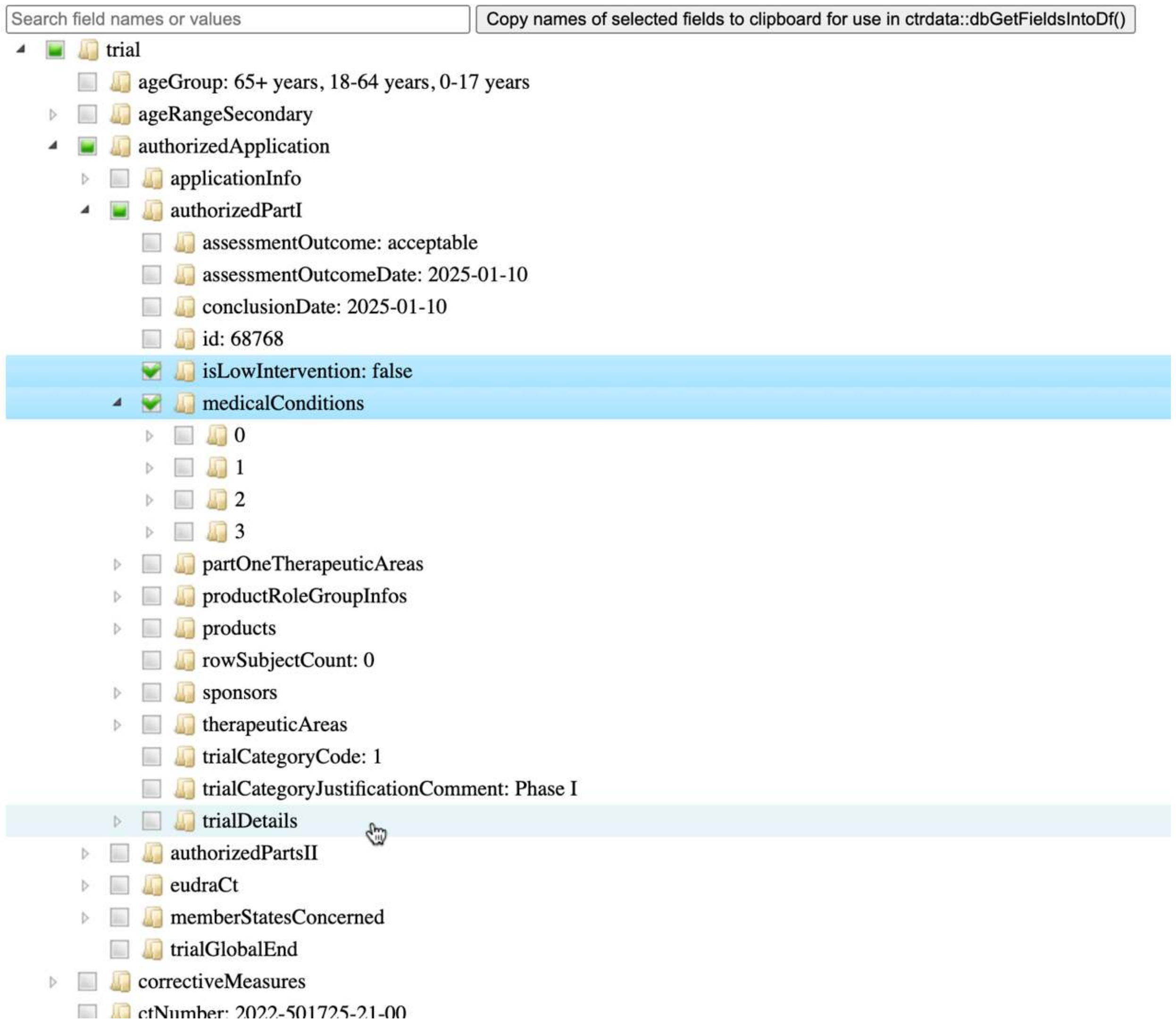
Alternatively, use any standard database function: Retrieve the trial's `JSON` representation that `ctrdata` had loaded into the database and visualise its nested structure of field names and values.

```
# Example requires a package for visualisation:
# remotes::install_github("hrbrmstr/jsonview")
#
# works with duckdb, SQLite, PostgreSQL, MongoDB
oneTrial <- nodbi::docdb_query(
  src = db,
  key = db$collection,
  query = '{"_id":"2022-501725-21-00"}',
  limit = 1L
)

# Interactive widget where nodes can be expanded:
jsonview::json_tree_view(oneTrial)
```

Simple analysis of dates

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



Search field names or values

Copy names of selected fields to clipboard for use in ctrdata::dbGetFieldsIntoDf()

```
trial
  ageGroup: 65+ years, 18-64 years, 0-17 years
  ageRangeSecondary
  authorizedApplication
    applicationInfo
    authorizedPartI
      assessmentOutcome: acceptable
      assessmentOutcomeDate: 2025-01-10
      conclusionDate: 2025-01-10
      id: 68768
      isLowIntervention: false
      medicalConditions
        0
        1
        2
        3
      partOneTherapeuticAreas
      productRoleGroupInfos
      products
      rowSubjectCount: 0
      sponsors
      therapeuticAreas
      trialCategoryCode: 1
      trialCategoryJustificationComment: Phase I
      trialDetails
    authorizedPartsII
    eudraCt
    memberStatesConcerned
    trialGlobalEnd
  correctiveMeasures
  ctNumber: 2022-501725-21-00
```

Figure 1: CtisNested

```
# Get data of interest
result <- dbGetFieldsIntoDf(
  fields = c("ctrname"),
  calculate = c(".isUniqueTrial", ".startDate"),
  con = db
)
# Querying database (33 fields)...
# - Finding duplicates among registers' and sponsor ids...
# - 151 EUCTR _id were not preferred EU Member State record for 58 trials
# - Keeping 235 / 46 / 0 / 0 / 26 records from CTGOV2 / EUCTR / CTGOV / ISRCTN / CTIS
# Calculating .startDate...

str(result)
# 'data.frame': 523 obs. of  4 variables:
#  $ _id           : chr  "2004-004386-15-DE" "2004-004386-15-ES" "2004-004386-15-GB" "2004-004386-15-I
#  $ ctrname       : chr  "EUCTR" "EUCTR" "EUCTR" "EUCTR" ...
#  $ .isUniqueTrial: logi  FALSE FALSE FALSE FALSE FALSE TRUE ...
#  $ .startDate    : Date, format: "2005-07-26" "2005-11-15" "2005-07-26" "2005-08-29" ...
```

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")

# De-duplicate and visualise start date
hist(
  result[result$.isUniqueTrial, ".startDate"],
  breaks = "years"
)
box()
dev.off()
```

Cross-register clinical trial concepts

A number of calculations across fields from different registers are available in `ctrdata`. This offers a high level of convenience for users and begins building a canonical understanding and implementation of a correspondence mapping between fields and their values or code lists from different registers. Available calculations can be listed, and a specific documentation is shown as follows.

```
# List functions to calculate trial concepts across registers:
dfCalculate()
#  [1] ".controlType"               ".isMedIntervTrial"        ".isPlatformTrial"
#  [4] ".isUniqueTrial"             ".numSites"                ".numTestArmsSubstances"
#  [7] ".primareEndpointEstimand"   ".sampleSize"              ".sponsorType"
# [10] ".startDate"                 ".statusRecruitment"       ".trialObjective"
# [13] ".trialPhase"

# For an overview, call:
help("ctrdata-trial-concepts")

# Show documentation of a specific calculation:
```

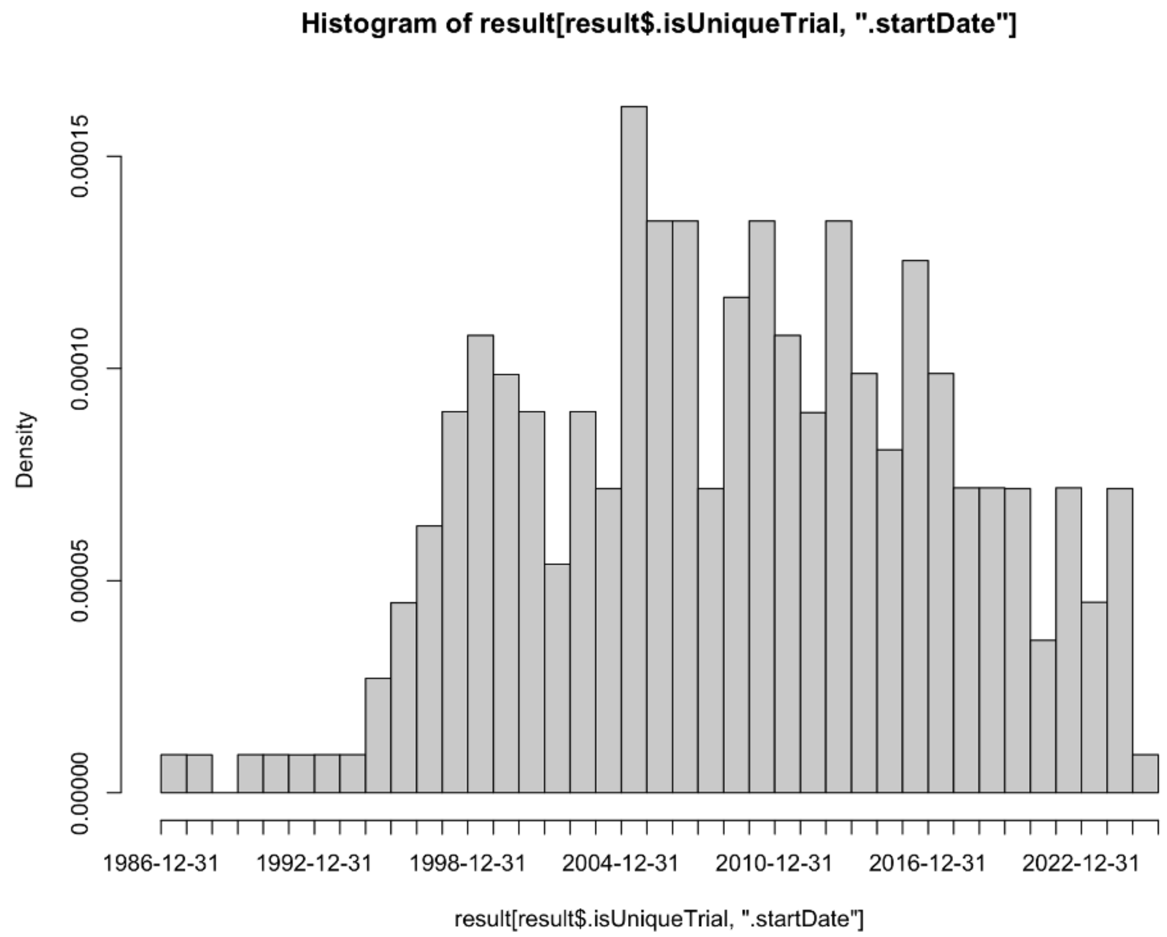


Figure 2: Histogram1

```
dfCalculate(name = ".isMedIntervTrial")
#
# >>>> .isMedIntervTrial
#
# * Description:
#
# Calculates if record is a medicine-interventional trial,
# investigating one or more medicine, whether biological or not.
#
# For EUCTR and CTIS, this corresponds to all records as per
# the definition of the EU Clinical Trial Regulation.
# For CTGOV and CTGOV2, this is based on drug or biological as
# type of intervention, and interventional as type of study.
# For ISRCTN, this is based on drug or biological as type of
# intervention, and interventional as type of study.
#
# Returns a logical.
#
# * Fields needed:
#
# ctrname
# intervention.intervention_type
# study_type
# protocolSection.armsInterventionsModule.interventions.type
# protocolSection.designModule.studyType
# interventions.intervention.interventionType
# trialDesign.primaryStudyDesign
# ctrname
#
# * To show the implementation, call: ctrdata::.isMedIntervTrial
# or, after package("ctrdata"), call: .isMedIntervTrial
#
# <<<<
```

Merging fields for analysis

With the function `dfMergeVariablesRelevel()`, users have control how to merge values of a set of original variables to a new variable, optionally with a new set of values. For more, see `help(dfMergeVariablesRelevel)`.

User annotations

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': 577 obs. of  3 variables:
#  $ _id               : chr  "2004-004386-15-DE" "2004-004386-15-ES" "2004-004386-15-GB" "2004-004386-
#  $ record_last_import: Date, format: "2025-02-15" "2025-02-15" "2025-02-15" ...
#  $ annotation        : chr  "test annotation" "test annotation" "test annotation" "test annotation" .
```

Analysing historic versions of trial records for changes in sample sizes

Historic versions can set to be retrieved for CTGOV2 by specifying `ctgov2history = <...>` when using `ctrLoadQueryIntoDb()`; this functionality was added in `ctrdata` version 1.18.0. The versions include all trial data available at the date of the respective version.

For CTGOV2 records, the historic versions are added as follows into the `ctrdata` data model of a trial record, where the ellipsis `...` represents all trial data fields:

`{"_id":"NCT01234567", "title": "Current title", ..., "history": [{"history_version": {"version_number": 1, "version_date": "2020-21-22 10:11:12"}, "title": "Original title", ...}, {"history_version": {"number": 2, "date": "2021-22-23 11:13:13"}, "title": "Later title", ...}]}`

The example shows how planned or realised number of participants (sample size) changed over time for individual trials, using available data (that is, only from `CTGOV2`; historic versions were available for `CTIS` only until its relaunch on 2024-06-17).

```
#
# load some trials from CTGOV2 specifying that
# for each trial, 5 versions should be retrieved
ctrLoadQueryIntoDb(
  queryterm = "cond=neuroblastoma&aggFilters=phase:3,status:com",
  register = "CTGOV2",
  con = db,
  ctgov2history = 5L
)
# * Appears specific for CTGOV REST API 2.0
# * Found search query from CTGOV2: cond=neuroblastoma&aggFilters=phase:3,status:com
# * Checking trials using CTGOV REST API 2.0, found 26 trials
# (1/3) Downloading in 1 batch(es) (max. 1000 trials each; estimate: 2.6 Mb total)
# Download status: 1 done; 0 in progress. Total size: 1.13 Mb (475%)... done!
# (2/3) Converting to NDJSON...
# (3/3) Importing records into database...
# JSON file #: 1 / 1
# * Checking and processing historic versions...
# Download status: 26 done; 0 in progress. Total size: 1.38 Mb (507%)... done!
# - Downloading 127 historic versions (estimate: 4.8 MB total)...
# Download status: 127 done; 0 in progress. Total size: 4.29 Mb (525%)... done!
# - Merging trial versions . . . . . . . . . . . . . . . . . . . . . . . . .
# - Updating trial records . . . . . . . . . . . . . . . . . . . . . . . . .
# Updated 26 trial(s) with historic versions
# = Imported or updated 26 trial(s)
# Updated history ("meta-info" in "my_collection_name")

# get relevant fields
result <- dbGetFieldsIntoDf(
  fields = c(
    # CTGOV2
    "history.protocolSection.designModule.enrollmentInfo.count",
    "history.history_version.version_date"
  ),
  con = db
)

# helpers
library(dplyr)
library(tidyr)
library(ggplot2)

# mangle and plot
result %>%
  unnest(cols = starts_with("history.")) %>%
  group_by(`_id`) %>%
  ggplot(
    mapping = aes(
      x = history.history_version.version_date,
      y = history.protocolSection.designModule.enrollmentInfo.count,
      colour = `_id`)
  ) +
  geom_step() +
  geom_point() +
  theme_light() +
  guides(colour = "none") +
  labs(
    title = "Sample sizes in trials including patients with a neuroblastoma",
    subtitle = "Source: CTGOV2 records labelled as phase 3 and completed",
    caption = Sys.Date()
  )

ggsave("vignettes/samplesizechanges.png", width = 6, height = 4)
```

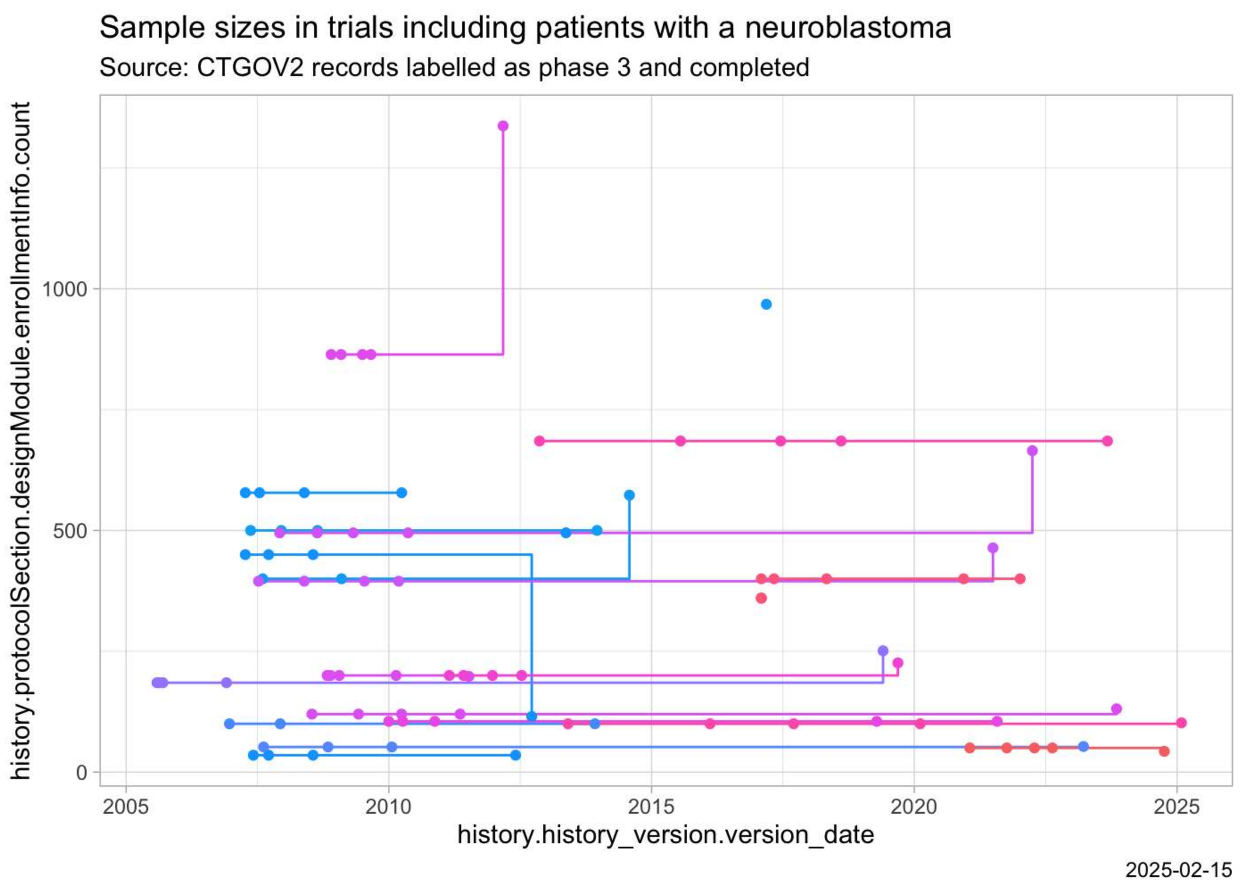
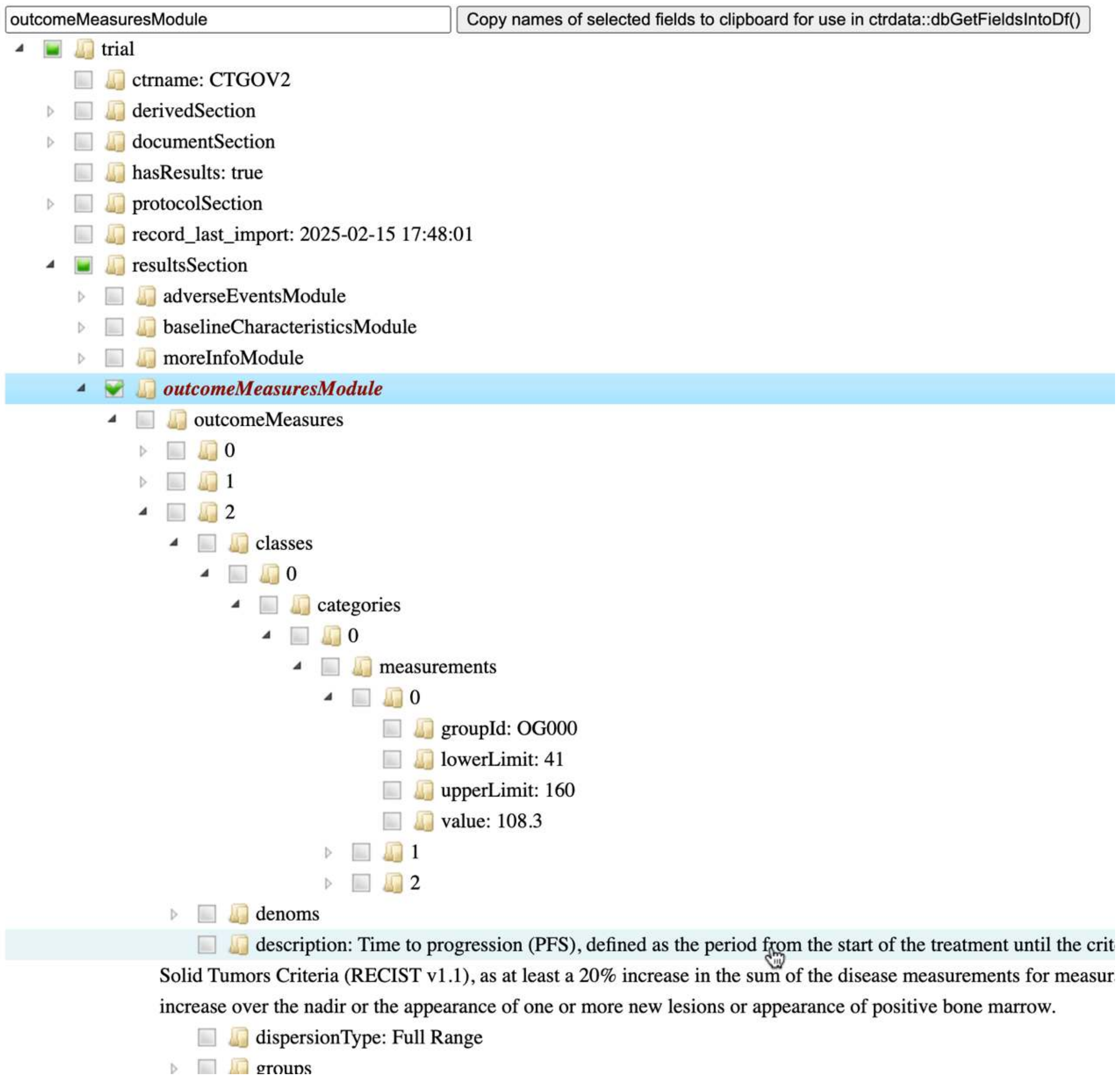


Figure 3: Sample size changes

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 CTGOV2 follows. Compare this with the outcome measures presented here: https://clinicaltrials.gov/study/NCT02139397?tab=results#outcome-measures, specifically "3. Determine the Progression Free Survival (PFS)..."

```r
#
# Since version 1.20.0, an interactive widget is built into ctrdata
# and can be used to search in all field names and all values
ctrShowOneTrial("NCT02139397", con = db)
```


outcomeMeasuresModule
Copy names of selected fields to clipboard for use in ctrdata::dbGetFieldsIntoDf()

```
trial
  ctrname: CTGOV2
  derivedSection
  documentSection
  hasResults: true
  protocolSection
  record_last_import: 2025-02-15 17:48:01
  resultsSection
    adverseEventsModule
    baselineCharacteristicsModule
    moreInfoModule
    outcomeMeasuresModule
      outcomeMeasures
        0
        1
        2
          classes
            0
              categories
                0
                  measurements
                    0
                      groupId: OG000
                      lowerLimit: 41
                      upperLimit: 160
                      value: 108.3
                    1
                    2
          denoms
          description: Time to progression (PFS), defined as the period from the start of the treatment until the crit
Solid Tumors Criteria (RECIST v1.1), as at least a 20% increase in the sum of the disease measurements for measur
increase over the nadir or the appearance of one or more new lesions or appearance of positive bone marrow.
          dispersionType: Full Range
          groups
```

Figure 4: Ctgov2NestedResults

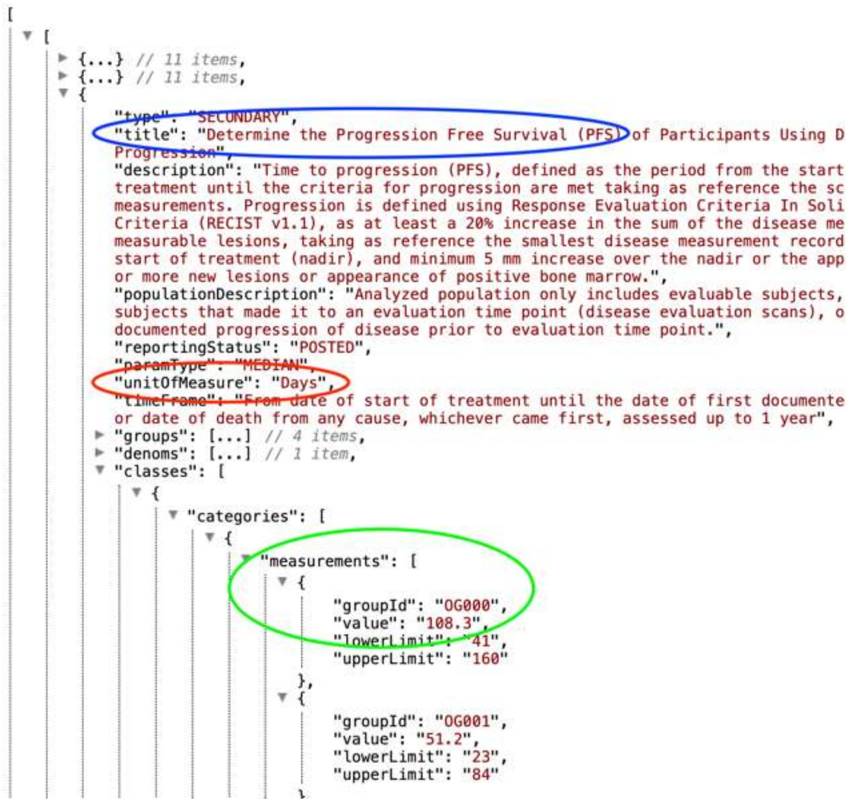
Alternative:

```r
# remotes::install_github("https://github.com/hrbrmstr/jsonview")

# Find top level nodes within resultsSection
dbFindFields("^resultsSection[.][^.]+$", db)
# Using cache of fields.
#                                         CTGOV2
#            "resultsSection.adverseEventsModule"
#                                         CTGOV2
# "resultsSection.baselineCharacteristicsModule"
#                                         CTGOV2
#                 "resultsSection.moreInfoModule"
#                                         CTGOV2
#          "resultsSection.outcomeMeasuresModule"
#                                         CTGOV2
#          "resultsSection.participantFlowModule"

# Get relevant data
result <- dbGetFieldsIntoDf("resultsSection.outcomeMeasuresModule", con = db)

# Create interactive widget
jsonview::json_tree_view(result[result[["_id"]] == "NCT02139397", -1])
```



```
[
  ▼ [
    ▶ {...} // 11 items,
    ▶ {...} // 11 items,
    ▼ {
        "type": "SECONDARY",
        "title": "Determine the Progression Free Survival (PFS) of Participants Using D
        Progression",
        "description": "Time to progression (PFS), defined as the period from the start
        treatment until the criteria for progression are met taking as reference the sc
        measurements. Progression is defined using Response Evaluation Criteria In Soli
        Criteria (RECIST v1.1), as at least a 20% increase in the sum of the disease me
        measurable lesions, taking as reference the smallest disease measurement record
        start of treatment (nadir), and minimum 5 mm increase over the nadir or the app
        or more new lesions or appearance of positive bone marrow.",
        "populationDescription": "Analyzed population only includes evaluable subjects,
        subjects that made it to an evaluation time point (disease evaluation scans), o
        documented progression of disease prior to evaluation time point.",
        "reportingStatus": "POSTED",
        "paramType": "MEDIAN",
        "unitOfMeasure": "Days",
        "timeFrame": "From date of start of treatment until the date of first documente
        or date of death from any cause, whichever came first, assessed up to 1 year",
      ▶ "groups": [...] // 4 items,
      ▶ "denoms": [...] // 1 item,
      ▼ "classes": [
          ▼ {
              ▼ "categories": [
                  ▼ {
                      ▼ "measurements": [
                          ▼ {
                                "groupId": "OG000",
                                "value": "108.3",
                                "lowerLimit": "41",
                                "upperLimit": "160"
                            },
                          ▼ {
                                "groupId": "OG001",
                                "value": "51.2",
                                "lowerLimit": "23",
                                "upperLimit": "84"
                            }
```

Figure 5: Ctgov2NestedResults

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

1. Create a data from fields identified as shown in previous sections (using `dbGetFieldsIntoDf()`)
2. Transform nested information to a long, name-value data frame (using `dfTrials2Long()`)
3. Identify the measures of interest (e.g. PFS, blue circle above) by specifying the name and value of these fields (`wherename`, `wherevalue` in function `dfName2Value()`) and
4. Obtain values by specifying the name(s) of its value field(s) (red and green circles in figure above; `valuename` in function `dfName2Value()`).

This is put together in the following example. Note that `CTGOV` fields are no longer downloadable (see NEWS.md) but may exist in previously created databases.

```r
#
#### 1. Create data frame from results fields
#
# These are key results fields from
# CTGOV2, CTGOV and from EUCTR:
result <- dbGetFieldsIntoDf(
  fields = c(
    # EUCTR - note this requires to set parameter
    # euctrresults = TRUE in ctrLoadQueryIntoDb()
    # as shown above in section "User annotations"
    "trialInformation.populationAgeGroup",
    "subjectDisposition.recruitmentDetails",
    "baselineCharacteristics.baselineReportingGroups.baselineReportingGroup",
    "endPoints.endPoint",
    "subjectAnalysisSets",
    "adverseEvents.seriousAdverseEvents.seriousAdverseEvent",
    # CTGOV2
    "resultsSection.outcomeMeasuresModule",
    "protocolSection.designModule.designInfo.allocation",
    "resultsSection.participantFlowModule",
    # 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"
  ),
  con = db
)

# Keep only unique trial records
result <- result[result[["_id"]] %in% dbFindIdsUniqueTrials(con = db), ]


#### 2. 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 128644 rows, 157 unique names of variables

# explore
```

```r
result <- dbGetFieldsIntoDf("endPoints", con = db)
jsonview::json_tree_view(result[result[["_id"]] == "2010-019348-37-IT", -1])

#### 3. Obtain values for measures of interest
#
# The parameters can be regular expressions.
clinicalDuration <- dfName2Value(
  df = long_result,
  wherename = paste0(
    "endPoints.endPoint.title|",
    "resultsSection.outcomeMeasuresModule.outcomeMeasures.title"
  ),
  wherevalue = paste0(
    "duration of response|DOR|",
    "free survival|DFS|PFS|EFS"
  ),
  valuename = paste0(
    "resultsSection.*outcomeMeasures.classes.categories.measurements.value|",
    "endPoints.*armReportingGroup.tendencyValues.tendencyValue.value|",
    "resultsSection.outcomeMeasuresModule.outcomeMeasures.unitOfMeasure|",
    "endPoints.endPoint.unit|",
    "resultsSection.outcomeMeasuresModule.outcomeMeasures.groups.title|",
    "endPoints.*armReportingGroup.armId"
  )
)
# Returning values for 50 out of 144 trials

# Duration has been reported with various units:
sort(unique(clinicalDuration[grepl("unit", clinicalDuration$name), "value", drop = TRUE]))

# For convenience, `dplyr` and related functions can be used, as follows:
library(dplyr)
library(tidyr)

clinicalDuration %>%
  as_tibble() %>%
  mutate(
    group_id = paste0(`_id`, "_", sub("([0-9]+)[.]?.*", "\\1", identifier)),
    name_short = sub(".*[.](.+)", "\\1", name),
    name_short = if_else(name_short == "unitOfMeasure", "unit", name_short)
  ) %>%
  group_by(group_id) %>%
  mutate(
    is_duration = any(grepl("day|month|week|year", value, ignore.case = TRUE))) %>%
  ungroup() %>%
  filter(is_duration) %>%
  select(name_short, value, where, group_id) %>%
  pivot_wider(id_cols = c(group_id, where), names_from = name_short, values_fn = list) %>%
  unnest(c(value, unit)) %>%
  filter(!grepl("999[9]*", value)) %>%
  rowwise() %>%
  mutate(
    value = as.numeric(value),
    arm_names = paste(armId, title, collapse = " / "),
  ) %>%
  ungroup() %>%
  mutate(
    days = case_when(
      grepl("[wW]eek", unit) ~ value * 7,
      grepl("[mM]onth", unit) ~ value * 30,
      grepl("[yY]ear", unit) ~ value * 30,
      .default = value
    )) %>%
  select(!c(value, unit, armId, title)) -> clinicalDuration

clinicalDuration
# # A tibble: 184 × 4
#    group_id              where                                                        arm_name
#    <chr>                 <chr>                                                        <chr>
#  1 2010-022951-49-LT_2  Progression-free Survival                                    "Arm-648
#  2 2010-022951-49-LT_2  Progression-free Survival                                    "Arm-648
#  3 2010-022951-49-LT_4  Progression Free Survival (PFS) in Participants with Wild-type RAS   "Arm-648
#  4 2010-022951-49-LT_4  Progression Free Survival (PFS) in Participants with Wild-type RAS   "Arm-648
#  5 2011-004168-30-BE_2  Progression-free survival PP population                      "Arm-937
#  6 2011-004168-30-BE_2  Progression-free survival PP population                      "Arm-937
#  7 2012-000510-10-GB_2  Progression Free Survival                                    "Arm-782
#  8 2012-004228-40-FR_4  Duration of Response (DOR)                                   "Arm-671
#  9 2013-003595-12-3RD_3 Progression Free Survival (PFS) as assessed by the Investigator by ... "Arm-1
# 10 2013-003595-12-3RD_3 Progression Free Survival (PFS) as assessed by the Investigator by ... "Arm-1
```

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.

```
#
result <- dbGetFieldsIntoDf(
  c(
    # CTGOV
    "resultsSection.outcomeMeasuresModule.outcomeMeasures.title",
    "protocolSection.designModule.enrollmentInfo.count",
    # EUCTR
    "e51_primary_end_points",
    "f11_number_of_subjects_for_this_age_range"
  ),
  con = db
)

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

```r
# Merge primary endpoint (pep)
result$pep <- dfMergeVariablesRelevel(
  df = result,
  colnames = c(
    "resultsSection.outcomeMeasuresModule.outcomeMeasures.title",
    "e51_primary_end_points"
  )
)

# Merge number of subjects
result$nsubj <- dfMergeVariablesRelevel(
  df = result,
  colnames = c(
    "protocolSection.designModule.enrollmentInfo.count",
    "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
#   233    38

# Plot
library(ggplot2)
ggplot(
  data = result,
  aes(
    x = nsubj,
    y = pep_is_efs
  )
) +
  geom_boxplot() +
  scale_x_log10()
# Warning message:
# Removed 25 rows containing non-finite values (`stat_boxplot()`).

ggsave("vignettes/boxpep.png", width = 6, height = 4)
```

Analysis methods and p values

```r
# helper
normalise_string <- function(x) {
```

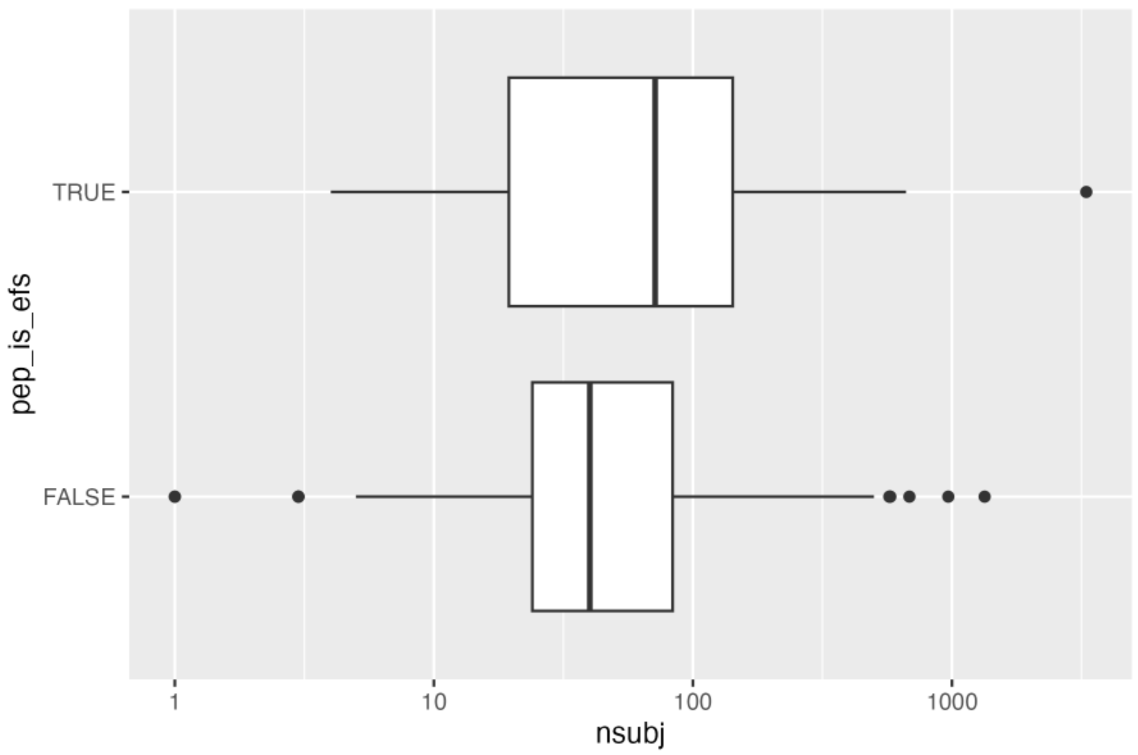


Figure 6: BoxPEP

```
  x <- gsub(",", "",  x)
  x <- gsub("-", " ", x)
  x <- tolower(x)
  x <- tools::toTitleCase(x)
  x <- gsub("[ ]+", " ", x)
  x <- trimws(x)
  x

}

# get trials into database
# about 5500 trials, loaded and important in less than 50 seconds
ctrLoadQueryIntoDb(
  queryterm = paste0(
    "https://clinicaltrials.gov/search?start=2010-01-01_2012-12-31", "
    &intr=Drug&aggFilters=ages:child,phase:2 3,results:with"),
  con = db)

# see definitions linked in help("ctrdate-registers") or directly go to
# https://clinicaltrials.gov/data-api/about-api/study-data-structure

# get result set
result <- dbGetFieldsIntoDf(c(
  "protocolSection.armsInterventionsModule.armGroups.type",
  "protocolSection.designModule.designInfo.allocation",
  "protocolSection.designModule.designInfo.interventionModel",
  "protocolSection.designModule.designInfo.maskingInfo.masking",
  "protocolSection.designModule.enrollmentInfo.count",
  "protocolSection.statusModule.startDateStruct.date",
  "protocolSection.outcomesModule.primaryOutcomes.measure",
  "resultsSection.outcomeMeasuresModule.outcomeMeasures.analyses.statisticalMethod",
  "resultsSection.outcomeMeasuresModule.outcomeMeasures.analyses.pValue"
),
con = db)

# # number of participants (last number is typically all groups summed up)
# # also see README.Rmd for alternative way, summing up non-total groups
# # result$totalparticipants <- vapply(
# # result[["clinical_results.baseline.analyzed_list.analyzed.count_list.count.value"]],
# # FUN = function(x) rev(x)[1], numeric(1L))
#
# View(result)
#
# result$totalparticipants <- result$protocolSection.designModule.enrollmentInfo.count

# first reported p value for primary endpoint analysis
result$pvalueprimaryanalysis <- vapply(
  result[["resultsSection.outcomeMeasuresModule.outcomeMeasures.analyses.pValue"]],
  FUN = function(x) as.numeric(trimws(gsub("[<>=]", "", strsplit(x, " / ")[[1]][1]))), numeric(1))

# statistical method used for primary endpoint analysis
result$methodprimaryanalysis <- vapply(
  result[["resultsSection.outcomeMeasuresModule.outcomeMeasures.analyses.statisticalMethod"]],
  FUN = function(x) normalise_string(strsplit(x, " / ")[[1]][1]), character(1))

# keep randomised, parallel-group, placebo-controlled, blinded clinical trials
result$prct <-
  grepl("PLACEBO|NO_INT", result$protocolSection.armsInterventionsModule.armGroups.type) &
  grepl("^RANDOM", result$protocolSection.designModule.designInfo.allocation) &
  grepl("^PARALLEL", result$protocolSection.designModule.designInfo.interventionModel) &
  !grepl("NONE", result$protocolSection.designModule.designInfo.maskingInfo.masking)
#
table(result$prct)
# FALSE   TRUE
#  4124   1561
#
result <- subset(result, prct == TRUE)

# helper
library(ggplot2)

# http://varianceexplained.org/statistics/interpreting-pvalue-histogram/
# http://www.pnas.org/content/100/16/9440.full
# plot p values
ggplot(
  result,
  aes(pvalueprimaryanalysis)) +
  stat_ecdf(geom = "step") +
  labs(
    title = paste0(
      "Paediatric phase 2 or 3 parallel-group interventional trials\n",
      "with randomisation to placebo or to no intervention"),
    x = "Range of p values",
    y = "Empirical cumulative density of p values\nof primary endpoint results") +
  geom_vline(
    xintercept = 0.05,
    linetype = 3)

ggsave("vignettes/phase23_paed_p_values.png", width = 6, height = 4)

# plot sample size v p value
ggplot(
  result,
  aes(
    x = protocolSection.designModule.enrollmentInfo.count,
    y = pvalueprimaryanalysis)) +
  geom_point() +
  ylim(0, 1) +
  xlim(0, 1000) +
  scale_x_log10() +
  geom_hline(yintercept = 0.05, linetype = 3)

ggsave("vignettes/phase23_paed_p_values_participants.png", width = 6, height = 4)

# statistical method used for primary endpoint analysis
tmp <- table(result$methodprimaryanalysis)
tmp <- tmp[rev(order(tmp))]
tmp <- data.frame(tmp)
knitr::kable(tmp[1:10,])
```

Var1	Freq
Ancova	226
Mixed Models Analysis	88
Cochran Mantel Haenszel	85
Anova	47
Fisher Exact	39
Chi Squared	37
t Test 2 Sided	34
log Rank	34
Regression Logistic	29
Wilcoxon (Mann Whitney)	27

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, CTGOV2 and ISRCTN is not known. The status is in the tree starting from the `dimp` element.

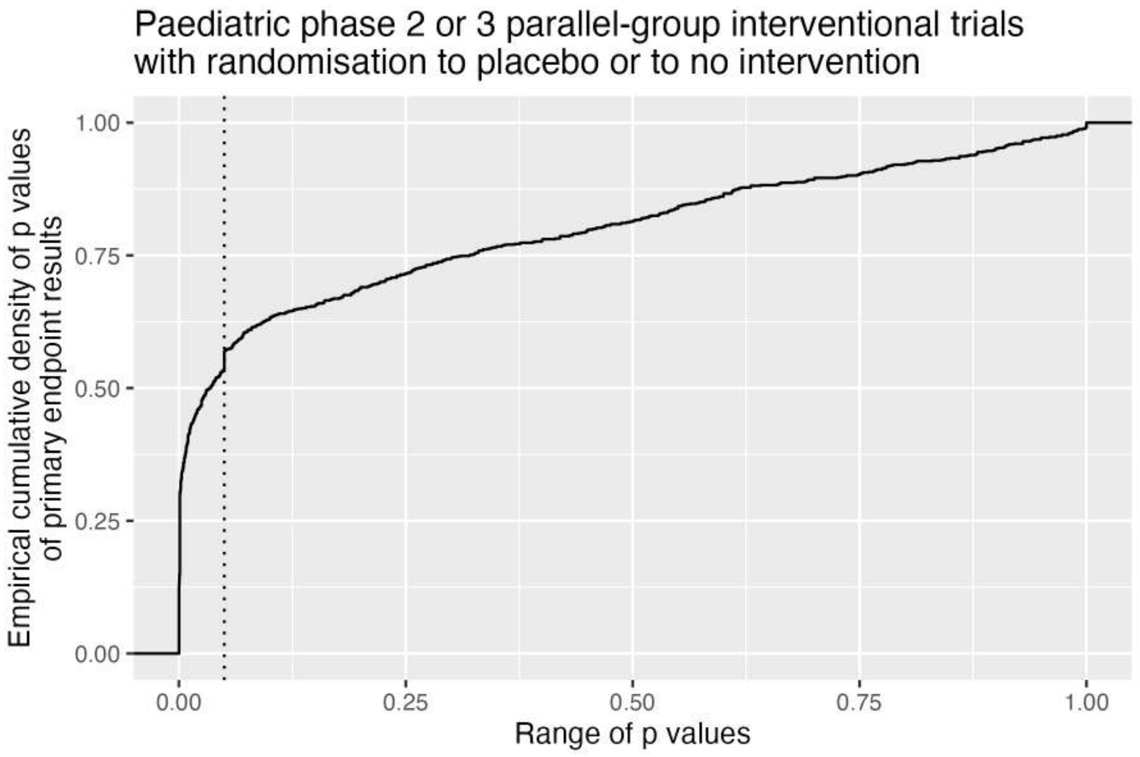


Figure 7: phase23_paed_p_values

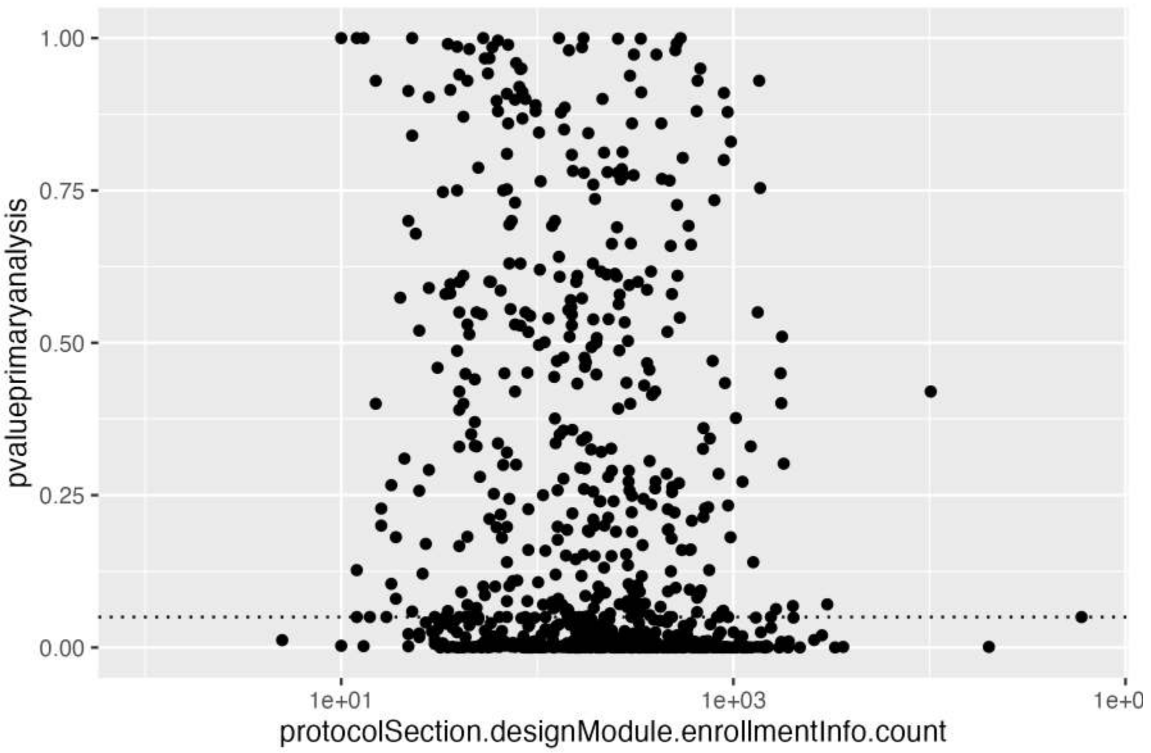


Figure 8: phase23_paed_p_values

```r
#
library(dplyr)

# 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 %>%
  rowwise() %>%
  mutate(startdatefirst = min(
    x, x6_date_on_which_this_record_was_first_entered_in_the_eudract_database, na.rm = TRUE)
  ) -> result

# 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  6  8 23
#  6  4  3  2  1  1  1
```

```r
result[["any_authorised"]] <- number_authorised > 0L

# Helper
library(ggplot2)
library(scales)

# Plot
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)
```

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.

```r
# 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] 6028

# 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] 342
```

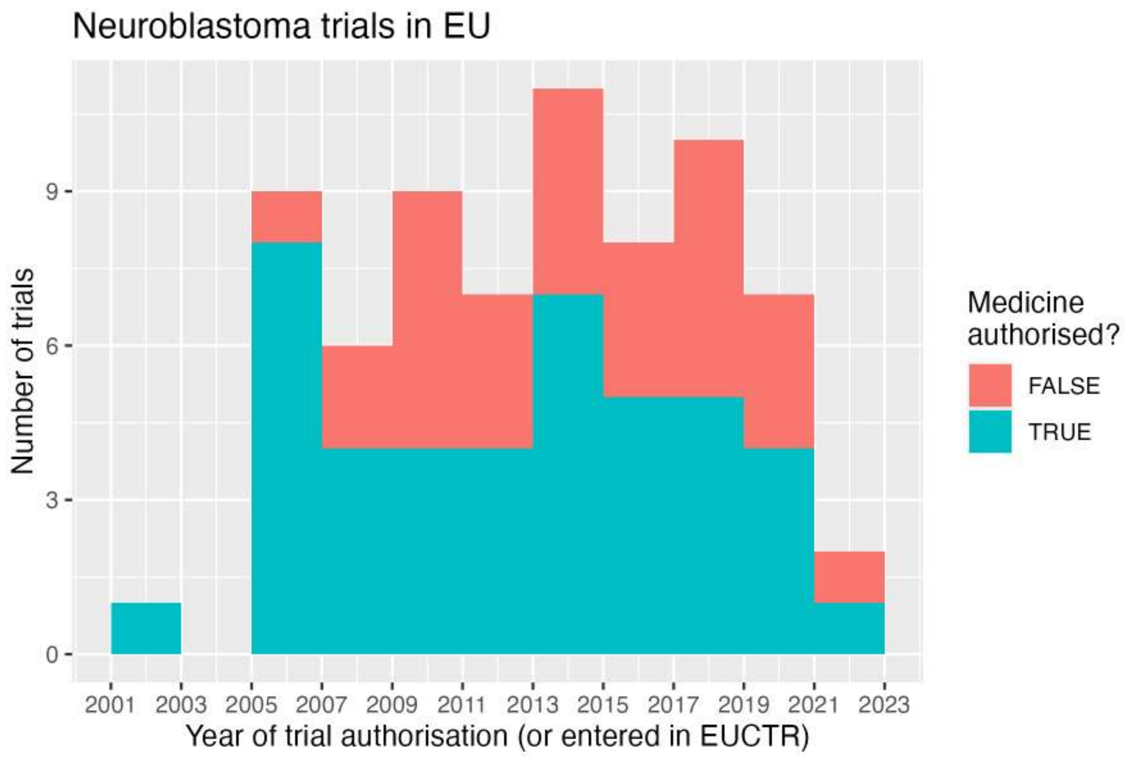


Figure 9: HistogramNBtrials

```
# Alternative:
m$count(query = '{"ctrname": "EUCTR"}')
# [1] 316

# Number of CTGOV records:
m$count(query = '{"_id": {"$regex": "NCT[0-9]{8}", "$options": "i"}}')
# [1] 5685

# Alternative:
m$count(query = '{"ctrname": "CTGOV2"}')
# [1] 5685

# To best define regular expressions for analyses, inspect the field:
head(
  m$distinct(
    # key = "resultsSection.outcomeMeasuresModule.outcomeMeasures.title",
    key = "protocolSection.outcomesModule.primaryOutcomes.measure",
    query = '{"ctrname": "CTGOV2"}'
  )
)
# [1] "\"Off\" Time"
# [2] "% Calories Taken Orally"
# [3] "% Change in Frequency of Handwaving Episodes"
# [4] "% of Patients in Whom a Sentinel Lymph Node is Identified"
# [5] "% of Subjects With Successful Preparation Rated by Colonoscopist on a 4 Point Scale (1=Poor to 4
# [6] "'Well-controlled Asthma Week' - a Derived Binary Variable (Yes/No)"
```

Aggregation

The following example uses the aggregation pipeline in MongoDB. See here for details on mongo's aggregation pipleline: 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": {"protocolSection.outcomesModule.primaryOutcomes.measure":
      {"$regex": "(progression|event|relapse|recurrence|disease)[- ]free",
                  "$options": "i"}}},
      {"$group": {"_id": "null", "count": {"$sum": 1}}}]'
)
out
#    _id count
# 1 null   282

# List records of trials with overall survival
# as primary endpoint, and list start date
out <- m$aggregate(
  pipeline =
    '[{"$match": {"protocolSection.outcomesModule.primaryOutcomes.measure":
      {"$regex": "overall survival", "$options": "i"}}},
      {"$project": {"_id": 1, "protocolSection.statusModule.startDateStruct.date": 1}}]'
)
head(out)
#           _id       date
# 1 NCT00793845    2008-08
# 2 NCT00027560    2001-07
# 3 NCT00515073    2001-04
# 4 NCT00050960    2002-05
# 5 NCT02659020 2016-03-01
# 6 NCT02151526 2013-06-07
```

Mapreduce

Since Mapreduce is deprecated starting in MongoDB 5 (https://docs.mongodb.com/manual/core/map-reduce/), use an aggregation pipeline:

```
# Count number of trials by number of study
# participants in bins of hundreds of participants:
m$aggregate(pipeline = '
[
  {
    "$project": {
      "flooredNumber": {
        "$multiply": [
          {
            "$floor": {
              "$divide": [
                {
                  "$toInt": "$protocolSection.designModule.enrollmentInfo.count"
                },
                100
              ]
            }
          },
          100
        ]
      }
    }
  },
  {
    "$group": {
      "_id": "$flooredNumber",
      "count": {
        "$count": {}
      }
    }
  },
  {
    "$sort": {
      "_id": 1
    }
  }
]
')
#      _id count
# 1     NA   370
# 2      0  3125
# 3    100   811
# 4    200   430
# 5    300   323
# 6    400   183
# 7    500   155
# 8    600   126
# 9    700    88
# 10   800    74
# 11   900    41
# [...]
```