EHR Vignette for Structured Data

2021-06-05

Introduction

The `EHR` package provides several modules to perform diverse medication-related studies using data from electronic health record (EHR) databases. Especially, the package includes modules to perform pharmacokinetic/pharmacodynamic (PK/PD) analyses using EHRs, as outlined in Choi *et al.*[1], and additional modules will be added in the future. This vignette describes four modules for processing data (*Pro-Demographic*, *Pro-Med-Str*, *Pro-Drug Level*, *Pro-Laboratory*) and one module for PK data building (*Build-PK-IV*) in the system, when data are typically obtained from a structured database.

The process starts with structured data extracted by Structured Query Language (SQL) from EHRs or provided by a user, then moves through two phases: data processing which standardizes and combines the input data (*Pro-Med-Str*, *Pro-Drug Level*, etc.) and data building which creates the final PK data (*Build-PK-IV*).

The vignette has two examples. The first example demonstrates how to build PK data quickly without using the data processing modules when cleaned data for concentration, drug dose, demographic and laboratory datasets are already available in an appropriate data form. The second example shows how to utilize several data processing modules to standardize and combine more complex datasets, and then finally build PK data using *Build-PK-IV*.

To begin we load the `EHR` package, the `pkdata` package, and the `lubridate` package.

```r
# load EHR package and dependencies
library(EHR)
library(pkdata)
library(lubridate)
```

Example 1: Quick Data Building with Processed Datasets

The data for example 1 includes a demographic file, a concentration file, an intravenous (IV) dosing file, and a laboratory file, which are all cleaned and formatted appropriately. We also define a directory for the raw data and a directory for interactive checking output files.

```r
# define directories
rawDataDir <- system.file("examples", "str_ex1", package="EHR")
checkDir <- system.file("examples", "str_ex1", "checks", package="EHR")

demo <- read.csv(file.path(rawDataDir,"Demographics_DATA_simple.csv"))
head(demo)
```

```
##   patient_id patient_visit_id gender weight height surgery_date ageatsurgery
## 1          2              2.1      1  62.99 179.72    6/20/2015         6245
## 2          3              3.1      0   7.71  72.99   12/15/2018          574
## 3          4              4.1      1  12.00  92.02    1/12/2018         1214
## 4          5              5.1      0   5.89  56.13    5/13/2018          102
## 5          6             6.2      1   7.90  63.31    9/24/2018          343
## 6          6             6.1      1   6.16  60.89    5/26/2018          222
##   stat_sts cpb_sts date_icu_dc time_fromor length_of_icu_stay
## 1        2      80   6/22/2015          NA                  2
## 2        3      67  12/16/2018          NA                  1
## 3        1      70   1/13/2018          NA                  1
## 4        1     151   5/14/2018        1427                  1
## 5        2      50   10/6/2018          NA                 12
## 6        4      99   8/13/2018        1958                 79
```

```
conc.data <- read.csv(file.path(rawDataDir,"Concentration_DATA_simple.csv"))
head(conc.data)
```

```
##   patient_id patient_visit_id event conc.level           date.time
## 1         10             10.1     4       0.17 2019-02-02 05:30:00
## 2         10             10.1     2       4.05 2019-02-24 14:00:00
## 3         10             10.1     3       0.64 2019-02-25 03:30:00
## 4         10             10.1     5       0.33 2019-02-27 02:45:00
## 5         10             10.1     6       0.07 2019-02-28 03:30:00
## 6         10             10.1     7       0.05 2019-03-01 02:35:00
```

```
ivdose.data <- read.csv(file.path(rawDataDir,"IVDose_DATA_simple.csv"))
head(ivdose.data)
```

```
##   patient_id  date.dose    infuse.time.real         infuse.time infuse.dose
## 1          1 2009-10-18 2009-10-18 11:35:00 2009-10-18 12:00:00         8.8
## 2          1 2009-10-18 2009-10-18 12:00:00 2009-10-18 12:00:00         8.8
## 3          1 2009-10-18 2009-10-18 13:00:00 2009-10-18 13:00:00         8.8
## 4          1 2009-10-18 2009-10-18 14:00:00 2009-10-18 14:00:00         8.8
## 5          1 2009-10-18 2009-10-18 15:00:00 2009-10-18 15:00:00         8.8
## 6          1 2009-10-18 2009-10-18 16:00:00 2009-10-18 16:00:00         8.8
##   bolus.time bolus.dose given.dose maxint weight
## 1       <NA>         NA          0     60    4.4
## 2       <NA>         NA          0     60    4.4
## 3       <NA>         NA          0     60    4.4
## 4       <NA>         NA          0     60    4.4
## 5       <NA>         NA          0     60    4.4
## 6       <NA>         NA          0     60    4.4
```

```
creat.data <- read.csv(file.path(rawDataDir,"Creatinine_DATA_simple.csv"))
head(creat.data)
```

```
##   patient_id           date.time creat
## 1          2 2015-06-23 04:35:00  0.75
## 2          2 2015-06-22 04:00:00  0.69
## 3          2 2015-06-21 01:55:00  0.78
## 4          2 2015-06-24 03:45:00  0.64
## 5          2 2015-06-14 15:11:00  0.71
## 6          2 2015-06-20 20:14:00  0.58
```

The `EHR` package modules use a standardized naming convention for patient identification (ID) variables. We rename the unique patient-level ID from `patient_id` to `mod_id` and the visit-level ID from `patient_visit_id` to `mod_id_visit`. If there is only a single visit/course per subject the unique patient-level ID and visit-level ID can be the same, however both `mod_id` and `mod_id_visit` should be defined.

```
names(conc.data)[1:2] <- names(demo)[1:2] <- c("mod_id", "mod_id_visit")
names(creat.data)[1] <- names(ivdose.data)[1] <- "mod_id"
```

Using the four datasets, we can build a final PK dataset with the function `run_Build_PK_IV()`. Additional details for this function are provided below in the *Build-PK-IV* subsection of Example 2: Complete Data Processing and Building from Raw Extracted Data to PK Data.

```r
simple_pk_dat <- run_Build_PK_IV(
    conc=conc.data,
    dose=ivdose.data,
    lab.dat = list(creat.data),
    lab.vars = c('creat'),
    demo.list = demo,
    demo.vars=c('weight', 'weight_demo', 'height', 'gender', 'ageatsurgery',
                'stat_sts', 'cpb_sts', 'length_of_icu_stay'),
    demo.abbr=c('wgt', 'wgt_demo', 'height', 'gender',
                'age', 'stat', 'cpb', 'loi'),
    pk.vars=c('mod_id_visit', 'time', 'conc', 'dose', 'rate', 'event',
              'other', 'multiple.record', 'date', 'mod_id'),
    drugname='fent',
    check.path=checkDir)
```

```
## 0 duplicated rows
## The dimension of the PK data before merging with demographics: 149 x 9
## The number of subjects in the PK data before merging with demographics: 10
## The number of subjects in the demographic file, who meet the exclusion
##      criteria: 0
## check NA frequency in demographics, see file /private/var/folders/06/
##      0qv1dr5508j_tbzqdjfqjf680000gn/T/RtmpxKeQ2k/Rinst31587fb28159/EHR/
##      examples/str_ex1/checks/fent-missing-demo.csv
## List of IDs missing at least 1 cpb_sts:
## Checked: all missing cpb_sts are 0
## The list of final demographic variables: weight
## weight_demo
## height
## gender
## ageatsurgery
## stat_sts
## cpb_sts
## length_of_icu_stay
## Checked: there are no missing creat
## The dimension of the final PK data exported with the key demographics:
##      149 x 16 with 10 distinct subjects (mod_id_visit)
```

```r
head(simple_pk_dat,15)
```

```
##    mod_id_visit  time conc amt rate mdv evid   wgt wgt_demo height gender  age
## 1           1.2  0.00   NA  50    0   1    1 25.04    25.04 114.39      1 2295
## 2           1.2  0.75   NA 100    0   1    1 25.04    25.04 114.39      1 2295
## 3           1.2  1.65   NA 100    0   1    1 25.04    25.04 114.39      1 2295
## 4           1.2  1.77   NA 250    0   1    1 25.04    25.04 114.39      1 2295
## 5           1.2  2.05   NA 250    0   1    1 25.04    25.04 114.39      1 2295
## 6           1.2  3.72   NA 250    0   1    1 25.04    25.04 114.39      1 2295
## 7           1.2  5.23   NA 100    0   1    1 25.04    25.04 114.39      1 2295
## 8           1.2  6.25 2.83  NA   NA   0    0 25.04    25.04 114.39      1 2295
## 9           1.2 20.68 0.41  NA   NA   0    0 25.04    25.04 114.39      1 2295
## 10          1.2 70.90 0.04  NA   NA   0    0 25.04    25.04 114.39      1 2295
## 11          1.2 95.25 0.01  NA   NA   0    0 25.04    25.04 114.39      1 2295
## 12          10.1  0.00   NA  25    0   1    1  6.06     6.06  63.20      0  209
## 13          10.1  0.52   NA 100    0   1    1  6.06     6.06  63.20      0  209
## 14          10.1  1.42   NA  25    0   1    1  6.06     6.06  63.20      0  209
## 15          10.1  2.77   NA  50    0   1    1  6.06     6.06  63.20      0  209
##    stat cpb loi creat
## 1     2  79   1  0.60
## 2     2  79   1  0.60
## 3     2  79   1  0.60
## 4     2  79   1  0.60
## 5     2  79   1  0.60
## 6     2  79   1  0.60
## 7     2  79   1  0.60
## 8     2  79   1  0.60
## 9     2  79   1  0.50
## 10    2  79   1  0.54
## 11    2  79   1  0.57
## 12    1 195   5  0.64
## 13    1 195   5  0.64
## 14    1 195   5  0.64
## 15    1 195   5  0.64
```

Example 2: Complete Data Processing and Building from Raw Extracted Data to PK Data

To begin example 2 we define three directories for the raw data, the processed data, and files used for interactive checking.

```
rawDataDir <- system.file("examples", "str_ex2", package="EHR")
dataDir <- system.file("examples", "str_ex2", package="EHR")
checkDir <- system.file("examples", "str_ex2", "checks", package="EHR")
```

Pre-Processing for Raw Extracted Data

The raw data for example 2 includes a demographic file for use with the *Pro-Demographic* module; two files for the *Pro-Drug Level* module; two dosing files for the *Pro-Med-Str* module; and two lab files for use with the *Pro-Laboratory* module.

The structured datasets extracted by SQL must go through a pre-processing stage which creates new ID variables and datasets that can be used by the data processing modules. The following annotated example demonstrates the three main steps of pre-processing: (1) read and clean raw data; (2) merge raw data to create new ID variables; (3) make new data for use with modules.

Each raw dataset should contain a subject unique ID, a subject visit ID, or both ids. In this example the subject unique ID is called `subject_uid` and the subject visit ID is called `subject_id`. The subject visit ID is a combination of subject and visit/course – e.g., `subject_id` 14.0 is the first course for subject 14, `subject_id` 14.1 is the second course for subject 14, and so on. `subject_uid` is a unique ID that is the same for all subject records. The integer part of `subject_id` has a 1-to-1 correspondence with `subject_uid` – for this example, `subject_uid` 62734832 is associated with both `subject_id` 14.0 and `subject_id` 14.1. If there is only a single visit/course per subject only the subject unique ID is needed.

(1) Read and clean raw data

The demographics data file contains ID variables `subject_id` and `subject_uid`, in addition to demographic variables such as gender, date of birth, height, weight, etc. The Demographics_DATA.csv file is read in using the `readTransform()` function.

```
# demographics data
demo.in <- readTransform(file.path(rawDataDir, "Demographics_DATA.csv"))
head(demo.in)
```

```
##   subject_id subject_uid gender weight height surgery_date ageatsurgery
## 1       1106    34364670      0   5.14  59.18    6/28/2014          141
## 2       1444    36792472      1   5.67  62.90    1/10/2016          292
## 3       1465    36292449      0  23.67 118.02    3/19/2016         2591
## 4       1520    34161967      0  14.07  97.04    7/18/2016         1320
## 5       1524    37857374      1  23.40 102.80    7/23/2016         1561
## 6       1550    37826262      1   6.21  62.03     9/4/2016          208
##   stat_sts cpb_sts in_hospital_mortality add_ecmo date_icu_dc time_fromor
## 1        3     133                     0        0    7/2/2014        1657
## 2        1      65                     0        0   1/12/2016        1325
## 3        2     357                     0        0   3/20/2016          NA
## 4        5      93                     0        0   7/19/2016        1745
## 5        3      87                     1        0   7/30/2016        1847
## 6        1     203                     0        0   9/11/2016        1210
```

The example concentration data consists of two files, SampleTimes_DATA.csv and SampleConcentration_DATA.csv containing the concentration sampling times and values, respectively.

The sampling times data csv file is read in with `read.csv()`. Then the function `dataTransformation()` is used to rename the variable `Study.ID` to `subject_id` and to create a new variable called `samp`, which indexes the sample number, using the `modify=` argument.

```
# concentration sampling times data
# read in raw data
samp.raw <- read.csv(file.path(rawDataDir, "SampleTimes_DATA.csv"))
head(samp.raw)
```

```
##   Study.ID Event.Name Sample.Collection.Date.and.Time
## 1    466.1   Sample 1                  2/3/2017 10:46
## 2    466.1   Sample 2                  2/4/2017 20:30
## 3   1106.0   Sample 1                 6/28/2014 13:40
## 4   1106.0   Sample 2                 6/29/2014 03:10
## 5   1106.0   Sample 3                 6/30/2014 03:35
## 6   1106.0   Sample 4                  7/1/2014 03:45
```

```
# transform data
samp.in0 <- dataTransformation(samp.raw,
    rename = c('Study.ID' = 'subject_id'),
    modify = list(samp = expression(as.numeric(sub('Sample ', '', Event.Name)))))
head(samp.in0)
```

```
##   subject_id Event.Name Sample.Collection.Date.and.Time samp
## 1      466.1   Sample 1                  2/3/2017 10:46    1
## 2      466.1   Sample 2                  2/4/2017 20:30    2
## 3     1106.0   Sample 1                 6/28/2014 13:40    1
## 4     1106.0   Sample 2                 6/29/2014 03:10    2
## 5     1106.0   Sample 3                 6/30/2014 03:35    3
## 6     1106.0   Sample 4                  7/1/2014 03:45    4
```

Equivalently, the function `readTransform()` can be used to read in and transform the data with a single function call.

```r
# read in and transform data
samp.in <- readTransform(file.path(rawDataDir, "SampleTimes_DATA.csv"),
    rename = c('Study.ID' = 'subject_id'),
    modify = list(samp = expression(as.numeric(sub('Sample ', '', Event.Name)))))
head(samp.in)
```

```
##   subject_id Event.Name Sample.Collection.Date.and.Time samp
## 1      466.1   Sample 1                  2/3/2017 10:46    1
## 2      466.1   Sample 2                  2/4/2017 20:30    2
## 3     1106.0   Sample 1                 6/28/2014 13:40    1
## 4     1106.0   Sample 2                 6/29/2014 03:10    2
## 5     1106.0   Sample 3                 6/30/2014 03:35    3
## 6     1106.0   Sample 4                  7/1/2014 03:45    4
```

The same steps can be used for the sample values data csv file. It is read in using `read.csv()`. Then using `dataTransformation()` the `subject_id` variable is created from the `name` variable using a call to the helper function `sampId()` in the `modify=` argument.

```r
# concentration sample values data
# read in raw data
conc.raw<-read.csv(file.path(rawDataDir, "SampleConcentration_DATA.csv"))
head(conc.raw)
```

```
##   record_id    name fentanyl_calc_conc
## 1         1 466.1_1         0.01413622
## 2         2 466.1_2         0.27982075
## 3         3  1106_1         6.11873679
## 4         4  1106_2         0.59161716
## 5         5  1106_3         0.11280471
## 6         6  1106_4         0.02112153
```

```r
# helper function used to make subject_id
sampId <- function(x) {
  # remove leading zeroes or trailing periods
  subid <- gsub('(^0*|\\.$)', '', x)
  # change _ to .
  gsub('_([0-9]+[_].*)$', '.\\1', subid)
}

# transform data
conc.in0 <- dataTransformation(conc.raw,
                    modify = list(
                    subid = expression(sampId(name)),
                    subject_id = expression(as.numeric(sub('[_].*', '', subid))),
                    samp = expression(sub('[^_]*[_]', '', subid)),
                    name = NULL,
                    data_file = NULL,
                    subid = NULL
                    )
                  )
head(conc.in0)
```

```
##   record_id fentanyl_calc_conc subject_id samp
## 1         1         0.01413622      466.1    1
## 2         2         0.27982075      466.1    2
## 3         3         6.11873679     1106.0    1
## 4         4         0.59161716     1106.0    2
## 5         5         0.11280471     1106.0    3
## 6         6         0.02112153     1106.0    4
```

Again, we can perform the same two steps with a single call to `readTransform()`.

```r
# equivalent using readTransform()
conc.in <- readTransform(file.path(rawDataDir, "SampleConcentration_DATA.csv"),
  modify = list(
    subid = expression(sampId(name)),
    subject_id = expression(as.numeric(sub('[_].*', '', subid))),
    samp = expression(sub('[^_]*[_]', '', subid)),
    name = NULL,
    data_file = NULL,
    subid = NULL
    )
  )
head(conc.in)
```

```
##   record_id fentanyl_calc_conc subject_id samp
## 1         1         0.01413622      466.1    1
## 2         2         0.27982075      466.1    2
## 3         3         6.11873679     1106.0    1
## 4         4         0.59161716     1106.0    2
## 5         5         0.11280471     1106.0    3
## 6         6         0.02112153     1106.0    4
```

The example drug dosing data consists of files FLOW_DATA.csv and MAR_DATA.csv containing two sources of IV dose information. The FLOW data csv file contains aliases for both ID variables; it is read in with the `readTransform()` function which renames the variables `Subject.Id` to `subject_id` and `Subject.Uniq.Id` to `subject_uid`.

```r
# FLOW dosing data
flow.in <- readTransform(file.path(rawDataDir, "FLOW_DATA.csv"),
                         rename = c('Subject.Id' = 'subject_id',
                                    'Subject.Uniq.Id' = 'subject_uid'))
head(flow.in)
```

```
##   subject_id subject_uid     Perform.Date FOCUS_MEDNAME Final.Wt..kg.
## 1       1596    38340814   12/4/2016 5:30      Fentanyl          6.75
## 2       1596    38340814   12/4/2016 6:00      Fentanyl          6.75
## 3       1596    38340814   12/4/2016 7:00      Fentanyl          6.75
## 4       1596    38340814   12/4/2016 7:40      Fentanyl          6.75
## 5       1607    38551767 12/24/2016 19:30      Fentanyl          2.60
## 6       1607    38551767 12/24/2016 20:00      Fentanyl          2.60
##   Final.Rate..NFR.units. Final.Units Flow
## 1            1 mcg/kg/hr       3.375   NA
## 2            1 mcg/kg/hr       6.750  0.1
## 3            1 mcg/kg/hr       4.500  0.1
## 4            0 mcg/kg/hr       0.000   NA
## 5            2 mcg/kg/hr       2.600   NA
## 6            2 mcg/kg/hr       5.200  0.2
```

The MAR data csv file contains several variables with a colon (:) character. To preserve the colon in these variable names, the data can be read in without checking for syntactically valid R variable names. The data is read in using `read.csv()` with the argument `check.names = FALSE` and then passed to the `dataTransformation()` function which renames `Uniq.Id` to `subject_uid`.

```
# MAR dosing data
mar.in0 <- read.csv(file.path(rawDataDir, "MAR_DATA.csv"), check.names = FALSE)
mar.in <- dataTransformation(mar.in0, rename = c('Uniq.Id' = 'subject_uid'))
head(mar.in)
```

```
##   subject_uid       Date  Time                 med:mDrug   med:dosage med:route
## 1    28579217 2017-02-04 19:15                Nicardipine 3 mcg/kg/min        IV
## 2    28579217 2011-10-02 22:11                 Famotidine       4.5 mg        IV
## 3    28579217 2011-10-02 20:17            Morphine sulfate         1 mg        IV
## 4    28579217 2011-10-03 02:28 Diphenhydramine injection        12 mg        IV
## 5    28579217 2011-10-02 22:11                  Cefazolin       225 mg        IV
## 6    28579217 2011-10-02 23:30            Morphine sulfate         1 mg        IV
##   med:freq med:given
## 1     <NA>     Given
## 2   q12hrs     Given
## 3  q2h prn     Given
## 4      now     Given
## 5    q8hrs     Given
## 6  q2h prn     Given
```

The example laboratory data consists of files Creatinine_DATA.csv and Albumin_DATA.csv. Both files are read in using the `readTransform()` function and `Subject.uniq` is renamed to `subject_uid`.

```
# Serum creatinine lab data
creat.in <- readTransform(file.path(rawDataDir, "Creatinine_DATA.csv"),
    rename = c('Subject.uniq' = 'subject_uid'))
head(creat.in)
```

```
##   subject_uid     date time creat
## 1    28579217 02/05/17 4:00  0.52
## 2    28579217 02/06/17 5:00  0.53
## 3    28579217 10/03/11 4:28  0.42
## 4    28579217 10/04/11 4:15  0.35
## 5    28579217 10/06/11 4:25  0.29
## 6    28579217 10/09/11 4:45  0.28
```

```
# Albumin lab data
alb.in <- readTransform(file.path(rawDataDir, "Albumin_DATA.csv"),
    rename = c('Subject.uniq' = 'subject_uid'))
head(creat.in)
```

```
##   subject_uid     date time creat
## 1    28579217 02/05/17 4:00  0.52
## 2    28579217 02/06/17 5:00  0.53
## 3    28579217 10/03/11 4:28  0.42
## 4    28579217 10/04/11 4:15  0.35
## 5    28579217 10/06/11 4:25  0.29
## 6    28579217 10/09/11 4:45  0.28
```

(2) Merge data to create new ID variables

The function `idCrosswalk()` merges all of the cleaned input datasets and creates new IDs. The `data=` argument of this function accepts a list of input datasets and the `idcols=` argument accepts a list of vectors or character strings that identify the ID variables in the corresponding input dataset.

The output of `idCrosswalk()` is a crosswalk dataset between the original ID variables (`subject_id`, `subject_uid`) and the new ID variables (`mod_id`, `mod_visit`, and `mod_id_visit`). The new variable `mod_id_visit` has a 1-to-1 correspondence to variable `subject_id` and uniquely identifies each subjects' visit/course; the new variable `mod_id` has a 1-to-1 correspondence to variable `subject_uid` and uniquely identifies each subject.

```
# merge all ID datasets
data <-  list(demo.in,
              samp.in,
              conc.in,
              flow.in,
              mar.in,
              creat.in,
              alb.in)

idcols <-  list(c('subject_id', 'subject_uid'), # id vars in demo.in
                'subject_id', # id var in samp.in
                'subject_id', # id var in conc.in
                c('subject_id', 'subject_uid'), # id vars in flow.in
                'subject_uid', # id var in mar.in
                'subject_uid', # id var in creat.in
                'subject_uid') # id var in creat.in

mod.id <- idCrosswalk(data, idcols, visit.id="subject_id", uniq.id="subject_uid")
saveRDS(mod.id, file=file.path(dataDir,"Fentanyl_module_id.rds"))

mod.id
```

```
##    subject_id subject_uid mod_visit mod_id mod_id_visit
## 1       466.0    28579217         1      1          1.1
## 2       466.1    28579217         2      1          1.2
## 3      1106.0    34364670         1      2          2.1
## 4      1444.0    36792472         1      3          3.1
## 5      1465.0    36292449         1      4          4.1
## 6      1520.0    34161967         1      5          5.1
## 7      1524.0    37857374         1      6          6.1
## 8      1550.0    37826262         1      7          7.1
## 9      1566.0    35885929         1      8          8.1
## 10     1566.1    35885929         2      8          8.2
## 11     1596.0    38340814         1      9          9.1
## 12     1607.0    38551767         1     10         10.1
## 13     1607.1    38551767         2     10         10.2
## 14     1724.0    39087607         1     11         11.1
## 15     1770.0    39418554         1     12         12.1
## 16     1770.1    39418554         2     12         12.2
## 17     2157.0    42023523         1     13         13.1
## 18     2162.0    42044808         1     14         14.1
## 19     2164.0    41221120         1     15         15.1
```

(3) Make new data for use with modules

The function `pullFakeId()` replaces the original IDs – `subject_id` and `subject_uid` – with new IDs – `mod_id`, `mod_visit`, and `mod_id_visit` – to create datasets which can be used by the data processing modules. Generally, the function call to `pullFakeId()` is

```
pullFakeId(dat, xwalk, firstCols = NULL, orderBy = NULL)
```

The `dat=` argument should contain the cleaned input data.frame from pre-processing step (1) and the `xwalk=` argument should contain the crosswalk data.frame produced in step (2). Additional arguments `firstCols=` and `orderBy=` control which variables are in the first columns of the output and the sort order, respectively. The cleaned structured data are saved as `R` objects for use with the modules.

```
## demographics data
demo.cln <- pullFakeId(demo.in, mod.id,
    firstCols = c('mod_id', 'mod_visit', 'mod_id_visit'),
    uniq.id = 'subject_uid')
head(demo.cln)
```

```
##   mod_id mod_visit mod_id_visit gender weight height surgery_date ageatsurgery
## 1      2         1          2.1      0   5.14  59.18    6/28/2014          141
## 2      3         1          3.1      1   5.67  62.90    1/10/2016          292
## 3      4         1          4.1      0  23.67 118.02    3/19/2016         2591
## 4      5         1          5.1      0  14.07  97.04    7/18/2016         1320
## 5      6         1          6.1      1  23.40 102.80    7/23/2016         1561
## 6      7         1          7.1      1   6.21  62.03     9/4/2016          208
##   stat_sts cpb_sts in_hospital_mortality add_ecmo date_icu_dc time_fromor
## 1        3     133                     0        0    7/2/2014        1657
## 2        1      65                     0        0   1/12/2016        1325
## 3        2     357                     0        0   3/20/2016          NA
## 4        5      93                     0        0   7/19/2016        1745
## 5        3      87                     1        0   7/30/2016        1847
## 6        1     203                     0        0   9/11/2016        1210
```

```
saveRDS(demo.cln, file=file.path(dataDir,"Fentanyl_demo_mod_id.rds"))

## drug level data
# sampling times
samp.cln <- pullFakeId(samp.in, mod.id,
    firstCols = c('mod_id', 'mod_visit', 'mod_id_visit', 'samp'),
    orderBy = c('mod_id_visit','samp'),
    uniq.id = 'subject_uid')
head(samp.cln)
```

```
##   mod_id mod_visit mod_id_visit samp Event.Name Sample.Collection.Date.and.Time
## 1      1         2          1.2    1   Sample 1                  2/3/2017 10:46
## 2      1         2          1.2    2   Sample 2                  2/4/2017 20:30
## 3     10         1         10.1    1   Sample 1                12/23/2016 05:15
## 4     10         1         10.1    2   Sample 2                12/24/2016 18:00
## 5     10         1         10.1    3   Sample 3                12/25/2016 03:00
## 6     10         1         10.1    4   Sample 4                12/26/2016 04:00
```

```
saveRDS(samp.cln, file=file.path(dataDir,"Fentanyl_samp_mod_id.rds"))

# sampling concentrations
conc.cln <- pullFakeId(conc.in, mod.id,
    firstCols = c('record_id', 'mod_id', 'mod_visit', 'mod_id_visit', 'samp'),
    orderBy = 'record_id',
    uniq.id = 'subject_uid')
head(conc.cln)
```

```
##   record_id mod_id mod_visit mod_id_visit samp fentanyl_calc_conc
## 1         1      1         2          1.2    1         0.01413622
## 2         2      1         2          1.2    2         0.27982075
## 3         3      2         1          2.1    1         6.11873679
## 4         4      2         1          2.1    2         0.59161716
## 5         5      2         1          2.1    3         0.11280471
## 6         6      2         1          2.1    4         0.02112153
```

```
saveRDS(conc.cln, file=file.path(dataDir,"Fentanyl_conc_mod_id.rds"))

## dosing data
# flow
flow.cln <- pullFakeId(flow.in, mod.id,
    firstCols = c('mod_id', 'mod_visit', 'mod_id_visit'),
    uniq.id = 'subject_uid')
head(flow.cln)
```

```
##   mod_id mod_visit mod_id_visit     Perform.Date FOCUS_MEDNAME Final.Wt..kg.
## 1      9         1          9.1   12/4/2016 5:30      Fentanyl          6.75
## 2      9         1          9.1   12/4/2016 6:00      Fentanyl          6.75
## 3      9         1          9.1   12/4/2016 7:00      Fentanyl          6.75
## 4      9         1          9.1   12/4/2016 7:40      Fentanyl          6.75
## 5     10         1         10.1 12/24/2016 19:30      Fentanyl          2.60
## 6     10         1         10.1 12/24/2016 20:00      Fentanyl          2.60
##   Final.Rate..NFR.units. Final.Units Flow
## 1            1 mcg/kg/hr       3.375   NA
## 2            1 mcg/kg/hr       6.750  0.1
## 3            1 mcg/kg/hr       4.500  0.1
## 4            0 mcg/kg/hr       0.000   NA
## 5            2 mcg/kg/hr       2.600   NA
## 6            2 mcg/kg/hr       5.200  0.2
```

```
saveRDS(flow.cln, file=file.path(dataDir,"Fentanyl_flow_mod_id.rds"))

# mar
mar.cln <- pullFakeId(mar.in, mod.id, firstCols = 'mod_id', uniq.id = 'subject_uid')
head(mar.cln)
```

```
##   mod_id       Date  Time                  med:mDrug   med:dosage med:route
## 1      1 2017-02-04 19:15                 Nicardipine 3 mcg/kg/min        IV
## 2      1 2011-10-02 22:11                  Famotidine       4.5 mg        IV
## 3      1 2011-10-02 20:17            Morphine sulfate         1 mg        IV
## 4      1 2011-10-03 02:28 Diphenhydramine injection        12 mg        IV
## 5      1 2011-10-02 22:11                   Cefazolin       225 mg        IV
## 6      1 2011-10-02 23:30            Morphine sulfate         1 mg        IV
##   med:freq med:given
## 1     <NA>     Given
## 2   q12hrs     Given
## 3  q2h prn     Given
## 4      now     Given
## 5    q8hrs     Given
## 6  q2h prn     Given
```

```
saveRDS(mar.cln, file=file.path(dataDir,"Fentanyl_mar_mod_id.rds"))

## laboratory data
```

```
creat.cln <- pullFakeId(creat.in, mod.id, 'mod_id',uniq.id = 'subject_uid')
head(creat.cln)
```

```
##   mod_id     date time creat
## 1      1 02/05/17 4:00  0.52
## 2      1 02/06/17 5:00  0.53
## 3      1 10/03/11 4:28  0.42
## 4      1 10/04/11 4:15  0.35
## 5      1 10/06/11 4:25  0.29
## 6      1 10/09/11 4:45  0.28
```

```
alb.cln <- pullFakeId(alb.in, mod.id, 'mod_id', uniq.id = 'subject_uid')
head(alb.cln)
```

```
##   mod_id     date  time alb
## 1      8 07/30/20  5:23 2.9
## 2      8 07/28/20  3:12 2.0
## 3      8 07/29/20  1:39 2.7
## 4      8 08/21/20 10:35 4.1
## 5      4 06/13/15 17:20 4.1
## 6      6 07/25/16  8:35 2.3
```

```
saveRDS(creat.cln, file=file.path(dataDir,"Fentanyl_creat_mod_id.rds"))
saveRDS(alb.cln, file=file.path(dataDir,"Fentanyl_alb_mod_id.rds"))
```

Before running the processing modules, it is necessary to define several options and parameters. Using `options(pkxwalk =)` allows the modules to access the crosswalk file. We also create a `drugname` stub and define the lower limit of quantification (LLOQ) for the drug of interest.

```
# set crosswalk option
xwalk <- readRDS(file.path(dataDir, "Fentanyl_module_id.rds"))
options(pkxwalk = 'xwalk')

# define parameters
drugname <- 'fent'
LLOQ <- 0.05
```

Pro-Demographic

The *Pro-Demographic* module accepts the cleaned structured demographic dataset and a user-defined set of exclusion criteria and returns a formatted list with the demographic data and records meeting the exclusion criteria suitable for integration with the other modules. For this example, we exclude subjects with a value of 1 for `in_hospital_mortality` or `add_ecmo` and create a new variable called `length_of_icu_stay`.

The demographic data can be processed by the `run_Demo()` function using:

```
# helper function
exclude_val <- function(x, val=1) { !is.na(x) & x == val }

demo.out <- run_Demo(demo.path = file.path(dataDir, "Fentanyl_demo_mod_id.rds"),
    toexclude = expression(exclude_val(in_hospital_mortality) | exclude_val(add_ecmo)),
    demo.mod.list = list(length_of_icu_stay =
                         expression(daysDiff(surgery_date, date_icu_dc))))
```

```
## The number of subjects in the demographic data, who meet the exclusion criteria: 2
```

```
head(demo.out$demo)
```

```
##   mod_id mod_visit mod_id_visit gender weight height surgery_date ageatsurgery
## 1      2         1          2.1      0   5.14  59.18    6/28/2014          141
## 2      3         1          3.1      1   5.67  62.90    1/10/2016          292
## 3      4         1          4.1      0  23.67 118.02    3/19/2016         2591
## 4      5         1          5.1      0  14.07  97.04    7/18/2016         1320
## 5      6         1          6.1      1  23.40 102.80    7/23/2016         1561
## 6      7         1          7.1      1   6.21  62.03     9/4/2016          208
##   stat_sts cpb_sts in_hospital_mortality add_ecmo date_icu_dc time_fromor
## 1        3     133                     0        0    7/2/2014        1657
## 2        1      65                     0        0   1/12/2016        1325
## 3        2     357                     0        0   3/20/2016          NA
## 4        5      93                     0        0   7/19/2016        1745
## 5        3      87                     1        0   7/30/2016        1847
## 6        1     203                     0        0   9/11/2016        1210
##   length_of_icu_stay
## 1                  4
## 2                  2
## 3                  1
## 4                  1
## 5                  7
## 6                  7
```

```
demo.out$exclude
```

```
## [1] "6.1"  "13.1"
```

- The `run_Demo` function arguments are as follows:
  - `demo.path`: file path where cleaned demographic data exist
  - `toexclude`: (optional) a set of user-defined expressions to set exclusion rules to be applied to the demographic dataset within the module
  - `demo.mod.list`: (optional) a list of user defined expressions to set modification rules within the module

See the `run_Demo()` function documentation for more details on the optional arguments.

- The output of `run_Demo()` is a list with two components: `demo`, a dataframe containing a demogrpahic information and `exclude`, vector of excluded visit IDs.

Pro-Med-Str

The *Pro-Med-Str* module processes structured medication data. Part I handles IV dose data and Part II handles e-prescription data.

Part I: IV dose data

Part I handles IV dose data from two sources, Flow data and Medication Administration Records (MAR) data. The Flow data are patient flow sheets which at this institution record infusion rates and changes to all infusions for all inpatients outside of the operating room. The MAR data record all bolus doses of medications and infusions administered in the operating room. The module is semi-interactive; it generates several files to check potential data errors and get feedback from an investigator. If corrected information ('fix' files) are provided, the module should be re-run to incorporate the corrections. The major functions of this module are:

- Process and clean Flow data by selecting, renaming, and modifying variables with the `flow.select`, `flow.rename`, and `flow.mod.list` parameters, and remove duplicate records, including invalid duplicate rows with `final.units` of 0, rate missing, one rate or unit missing, one unit of 0, or additional discrepancy.
- Process MAR data using the given medications in the `medchk.path` file, infusion units, bolus units, and bolus rate threshold. If there are data rows containing units other than those specified, a file listing these records will be produced using the `failunit_fn` stub appended with drugname.
  - e.g., if failunit_fn is 'Unit' the file 'failUnit-fent.csv' will be created in the `check.path` directory. The corrected file should replace 'fail' with 'fix' (e.g., 'fixUnit-fent.csv'), and include an additional variable called 'flag' with the value 'keep' for the records to be retained.
- Check for records with a unit containing 'kg' (e.g., 'mcg/kg/hr') but missing weight in flow data. These records will result in a calculated rate of NA when producing standardized rates (see step below). If there are records with missing weight, a file listing these records will be produced using the `failnowgt_fn` stub appended with drugname. A 'fix' file can be added to correct these records.
- Combine Flow and MAR infusion data and produce standardized rate (dose per unit time). Infusion rates with dose per weight per unit time (e.g., 'mcg/kg/hr') and dose per unit time ('mcg/hr') can both be handled. For infusion rates of dose per weight per unit time, weight is imputed using the closest available Flow weight (if possible) and the rate is multiplied by weight to get dose per unit time. Infusion rates of dose per unit time are assumed to be formatted correctly.
- Merge infusion dose data and bolus dose data and conform doses. See the `pkdata::conformDoses()` documentation for additional details.

The IV dose data can be processed by the `run_MedStrI()` function using:

```r
ivdose.out <- run_MedStrI(flow.path=file.path(dataDir,"Fentanyl_flow_mod_id.rds"),
    flow.select = c('mod_id','mod_id_visit','Perform.Date','Final.Wt..kg.',
                    'Final.Rate..NFR.units.','Final.Units'),
    flow.rename = c('mod_id','mod_id_visit', 'Perform.Date', 'weight',
                    'rate', 'final.units'),
    flow.mod.list = list(
      date.time = expression(parse_dates(fixDates(Perform.Date))),
      unit = expression(sub('.*[ ]', '', rate)),
      rate = expression(as.numeric(sub('([0-9.]+).*', '\\1', rate)))),
    medchk.path=file.path(rawDataDir, sprintf('medChecked-%s.csv', drugname)),
    mar.path=file.path(dataDir,"Fentanyl_mar_mod_id.rds"),
    demo.list=demo.out,
    check.path=checkDir,
    failflow_fn = 'FailFlow',
    failunit_fn = 'Unit',
    failnowgt_fn = 'NoWgt',
    infusion.unit = 'mcg/kg/hr',
    bolus.unit = 'mcg',
    bol.rate.thresh = Inf,
    drugname = drugname)
```

```
## The number of rows in the original data                124
## The number of rows after removing the duplicates       124
## no units other than mcg/kg/hr or mcg, file /private/var/folders/06/
##      0qv1dr5508j_tbzqdjfqjf680000gn/T/RtmpxKeQ2k/Rinst31587fb28159/EHR/
##      examples/str_ex2/checks/failUnit-fent.csv not created
## #########################
## 33 rows from 1 subjects with "kg" in infusion unit but missing weight,
##      see file /private/var/folders/06/0qv1dr5508j_tbzqdjfqjf680000gn/T/
##      RtmpxKeQ2k/Rinst31587fb28159/EHR/examples/str_ex2/checks/
##      failNoWgt-fent.csv AND create /private/var/folders/06/
##      0qv1dr5508j_tbzqdjfqjf680000gn/T/RtmpxKeQ2k/Rinst31587fb28159/EHR/
##      examples/str_ex2/checks/fixNoWgt-fent.csv
## #########################
```

```r
head(ivdose.out)
```

```
##   mod_id  date.dose infuse.time.real infuse.time infuse.dose
## 1      1 2011-10-02             <NA>        <NA>          NA
## 2      1 2011-10-02             <NA>        <NA>          NA
## 3      1 2017-02-04             <NA>        <NA>          NA
## 4      1 2017-02-04             <NA>        <NA>          NA
## 5      1 2017-02-04             <NA>        <NA>          NA
## 6      2 2014-06-28             <NA>        <NA>          NA
##            bolus.time bolus.dose given.dose maxint weight
## 1 2011-10-02 15:35:00         25         NA      0     NA
## 2 2011-10-02 17:26:00         25         NA      0     NA
## 3 2017-02-04 16:15:00         50         NA      0     NA
## 4 2017-02-04 16:30:00         20         NA      0     NA
## 5 2017-02-04 20:57:00         20         NA      0     NA
## 6 2014-06-28 08:15:00         20         NA      0     NA
```

- The `run_MedStrI` function arguments are as follows:
  - `flow.path`: file path where Flow data exist
  - `flow.select`: list of variables in the Flow data, which are used for processing
  - `flow.rename`: rename variables for the variables in flow.select
  - `flow.mod.list`: list containing modifications to variables in the Flow data
  - `medchk.path`: file path with list of medication names and variants in MAR data to be used
  - `mar.path`: file path where the MAR data exist
  - `demo.list`: (optional) file name for processed demographic file that is used to exclude subjects based on exclusion criteria
  - `check.path`: file path where the generated files for data checking are stored, and the corresponding data files with fixed data exist
  - `failflow_fn`: filename stub for invalid duplicate rows in the Flow data with rate of 0 check file. The stub will be prepended with the string 'fail' to create a .csv with the invalida data. The corrected data with failures replaced should use the same filename stub prepended with the string 'fix'.
    - e.g., if `failflow_fn` is 'FailFlow', the file 'failFailFlow.csv' with invalid duplicate rows will be created in the directory specified by `check.path`. The corrected version named 'fixFailFlow.csv' should be placed in the same directory.
  - `failunit_fn`: filename stub for records with units other than those specified with `infusion.unit` and `bolus.unit`
  - `failnowgt_fn`: filename stub for records with missing weight in the Flow data and unit involving 'kg'
  - `infusion.unit`: string specifying units for infusion doses (default: 'mcg/kg/hr')
  - `bolus.unit`: string specifying units for bolus doses (default: 'mcg')
  - `bol.rate.thresh`: upper bound for retaining bolus doses. Bolus units with a rate above the threshold are dropped (default: Inf; i.e., keep all bolus doses)
  - `drugname`: drug name of interest (e.g., dex, fent)
- The output of the `run_MedStrI` function is a dataset with processed IV dosing data including date (date.dose), infusion dose time (infuse.time.real and infuse.time) and rate (infuse.dose), bolus dose time (bolus.time) and dose level (bolus.dose), weight used in dose calculation and identification numbers for further merging with the output from other modules.

Part II: e-prescription data

Part II handles e-prescription data. To use this module, all prescriptions must be for only one drug. Different names, such as brand names and generic names, for the same drug are allowed (e.g., Lamictal and lamotrigine). The data used in this module must include columns for ID, date, strength, dose amount, and frequency. The major tasks the module performs are as follows:

- Creating numeric variables for strength, dose, and frequency
- Calculating daily dose
- Removing duplicate daily doses for a patient

There are two underlying functions used in this module. `processErx` performs the basic cleaning described above. `processErxAddl` performs some additional processing for more complicated dose expressions.

Below is example e-prescription data including columns for ID, drug name, dose, frequency, date, strength, and description.

```
(eRX <- read.csv(file.path(rawDataDir,"e-rx_DATA.csv"),stringsAsFactors = FALSE))
```

```
##   GRID    MED_NAME   RX_DOSE             FREQUENCY ENTRY_DATE STRENGTH_AMOUNT
## 1  ID1 lamotrigine         1                   bid 2009-02-24             100
## 2  ID2 lamotrigine         2                   bid 2006-12-30             100
## 3  ID2    Lamictal         1                   bid 2006-12-30             200
## 4  ID3 Lamictal XR         3                   bid 2004-08-24
## 5  ID4 lamotrigine         1           twice a day 2010-05-22          200 mg
## 6  ID5 lamotrigine    2 tabs                   qam 2007-06-13             200
## 7  ID6 lamotrigine 1.5+1+1.5 brkfst, lunch, dinner 2015-03-14             100
##                                          DESCRIPTION
## 1 lamotrigine 100 mg tablet (Also Known As Lamictal)
## 2 lamotrigine 100 mg tablet (Also Known As Lamictal)
## 3                              LaMICtal 200 mg tablet
## 4                        Lamictal XR 100 mg 24 hr Tab
## 5 lamotrigine 200 mg tablet (Also Known As Lamictal)
## 6         LaMICtal XR 200 mg tablet,extended release
## 7 lamoTRIgine 100 mg tablet (Also Known As Lamictal)
```

The e-prescription data can be processed by the `run_MedStrII` function using:

```
eRX.out <- run_MedStrII(file.path(rawDataDir,"e-rx_DATA.csv"),
    select = c('GRID','MED_NAME','RX_DOSE','FREQUENCY','ENTRY_DATE','STRENGTH_AMOUNT','DESCRIPTION'),
    rename = c('ID','MED_NAME','RX_DOSE','FREQUENCY','ENTRY_DATE','STRENGTH_AMOUNT','DESCRIPTION'))

eRX.out
```

```
##    ID    MED_NAME   RX_DOSE             FREQUENCY ENTRY_DATE STRENGTH_AMOUNT
## 1 ID1 lamotrigine         1                   bid 2009-02-24             100
## 2 ID2 lamotrigine         2                   bid 2006-12-30             100
## 4 ID3 Lamictal XR         3                   bid 2004-08-24
## 5 ID4 lamotrigine         1           twice a day 2010-05-22          200 mg
## 6 ID5 lamotrigine    2 tabs                   qam 2007-06-13             200
## 7 ID6 lamotrigine 1.5+1+1.5 brkfst, lunch, dinner 2015-03-14             100
##                                          DESCRIPTION strength freq.standard
## 1 lamotrigine 100 mg tablet (also known as lamictal)      100           bid
## 2 lamotrigine 100 mg tablet (also known as lamictal)      100           bid
## 4                        lamictal xr 100 mg 24 hr tab      100           bid
## 5 lamotrigine 200 mg tablet (also known as lamictal)      200           bid
## 6         lamictal xr 200 mg tablet,extended release      200            am
## 7 lamotrigine 100 mg tablet (also known as lamictal)      100           tid
##   freq.num dose daily.dose       date num_doses num_freqs
## 1        2    1        200 2009-02-24        NA        NA
## 2        2    2        400 2006-12-30        NA        NA
## 4        2    3        600 2004-08-24        NA        NA
## 5        2    1        400 2010-05-22        NA        NA
## 6        1    2        400 2007-06-13        NA        NA
## 7        3    4        400 2015-03-14         3         3
```

The following arguments are used in the `run_MedStrII` function:

- `file`: file name of prescription data
- `select`: the names of the columns to select
- `rename`: new column names; the default are the names required for the underlying functions, `processErx` and `processErxAddl`

In the above example, daily dose was calculated for the first 5 patients by multiplying strength*dose*freq.num, and a redundant daily dose was removed for the patient with ID2. In order to calculate a daily dose for the patient with ID3, the strength of 100 from the description was used because STRENGTH_AMOUNT was missing. For the patient with ID6, the dose amounts of 1.5, 1, and 1.5 are added together to get a dose of 4, and the daily dose is calculated as strength*dose.

Pro-Drug Level

Pro-Drug Level module processes drug concentration data that can be merged with medication dose data and other types of data. This module is semi-interactive; it generates several files while processing in order to check missing data and potential data errors, and get feedback from an investigator. If corrected information ('fix' files) are provided, the module should be re-run to incorporate the corrections. The major functions of this module are:

- Combine drug concentration data with sampling time: This step is necessary only if the drug concentration data file does not contain the sampling time (i.e., the time when blood samples were drawn for drug concentration measurements). When this is the case, the sampling time should be obtained from a separate data file.
- Check missing date/time for drug level measurements. If there are any missing dates, a file listing these records will be produced using the `failmiss_fn` stub appended with drugname. A 'fix' file can be added to correct these records. If no 'fix' file is provided, records with missing dates will be removed.
  - e.g., if `failmiss_fn` is 'MissingConcDate-', the file 'failMissingConcDate-dex.csv' will be created in the `check.path` directory. The file with corrected dates and times should replace 'fail' with 'fix' (e.g., 'fixMissingConcDate-dex.csv'), and be placed in the same `check.path` directory.
- Check for multiple sets of concentration data for the same subject, and keep the set of concentration data with the most drug concentration records above a lower limit of quantification (LLOQ). Data for subjects with multiple records, if any, is included in a file produced with the `multsets_fn` stub appended with the drugname and date.
  - e.g., if `multsets_fn` is 'multipleSetsConc-', the file 'multipleSetsConc-dex2020-07-15.csv' will be created in the `check.path` directory.
- Check duplicate concentration records on the same date/time. If there are any records with multiple concentrations on the same date/time, a file listing these records will be produced using the `faildup_fn` stub appended with drugname. A 'fix' file can be added to correct these records. If no 'fix' file is found, all duplicates are retained.
  - e.g., if `faildup_fn` is 'DuplicateConc-', the file 'failDuplicateConc-dex.csv' will be created in the `check.path` directory. The corrected file should replace 'fail' with 'fix', e.g. 'fixDuplicateConc-

dex.csv', and include an additional variable called 'flag' with the value 'keep' for the records to be retained.

The drug concentration data can be processed by the `run_DrugLevel` function using:

```
conc.out <- run_DrugLevel(conc.path=file.path(dataDir,"Fentanyl_conc_mod_id.rds"),
    conc.select=c('mod_id','mod_id_visit','samp','fentanyl_calc_conc'),
    conc.rename=c(fentanyl_calc_conc = 'conc.level', samp= 'event'),
    conc.mod.list=list(mod_id_event = expression(paste(mod_id_visit, event, sep = '_'))),
    samp.path=file.path(dataDir,"Fentanyl_samp_mod_id.rds"),
    samp.mod.list=list(mod_id_event = expression(paste(mod_id_visit, samp, sep = '_'))),
    check.path=checkDir,
    failmiss_fn = 'MissingConcDate-',
    multsets_fn = 'multipleSetsConc-',
    faildup_fn = 'DuplicateConc-',
    drugname=drugname,
    LLOQ=LLOQ,
    demo.list=demo.out)
head(conc.out)
```

```
## ##########################
## 3 rows need review, see file /private/var/folders/06/
##      0qv1dr5508j_tbzqdjfqjf680000gn/T/RtmpxKeQ2k/Rinst31587fb28159/EHR/
##      examples/str_ex2/checks/failMissingConcDate-fent.csv AND create /private/
##      var/folders/06/0qv1dr5508j_tbzqdjfqjf680000gn/T/RtmpxKeQ2k/
##      Rinst31587fb28159/EHR/examples/str_ex2/checks/fixMissingConcDate-fent.csv
## ##########################
## subjects with concentration missing from sample file
##  mod_id mod_id_event
##       8        8.1_1
##       8        8.1_2
##       8        8.1_3
## 1 subjects have multiple sets of concentration data
## 16 total unique subjects ids (including multiple visits) currently in the
##      concentration data
## 15 total unique subjects in the concentration data
## ##########################
## 15 rows need review, see file /private/var/folders/06/
##      0qv1dr5508j_tbzqdjfqjf680000gn/T/RtmpxKeQ2k/Rinst31587fb28159/EHR/
##      examples/str_ex2/checks/multipleSetsConc-fent2021-06-05.csv
## ##########################
## 15 total unique subjects ids (after excluding multiple visits) in the
##      concentration data
## 15 total unique subjects in the concentration data
```

```
head(conc.out)
```

```
##    mod_id mod_id_visit event  conc.level mod_id_event           date.time eid
## 1       1          1.2     1 0.014136220        1.2_1 2017-02-03 10:46:00   1
## 2       1          1.2     2 0.279820752        1.2_2 2017-02-04 20:30:00   1
## 55     10         10.1     2 3.136047304       10.1_2 2016-12-24 18:00:00   1
## 56     10         10.1     9 0.004720171       10.1_9 2017-01-01 04:20:00   1
## 57     10         10.1    10 0.017136367      10.1_10 2017-01-02 04:42:00   1
## 58     10         10.1    12 0.006335571      10.1_12 2017-01-04 03:40:00   1
```

The output provides a message that 3 rows are missing concentration date. The file 'failMissingConcDate-fent.csv' contains the 3 records with missing values for the `date.time` variable.

```r
( fail.miss.conc.date <- read.csv(file.path(checkDir,"failMissingConcDate-fent.csv")) )
```

```
##   subject_id subject_uid mod_id_event date.time
## 1       1566    35885929        8.1_1        NA
## 2       1566    35885929        8.1_2        NA
## 3       1566    35885929        8.1_3        NA
```

We can correct the missing dates by providing an updated file called 'fixMissingConcDate-fent.csv' that contains the missing data.

```r
fail.miss.conc.date[,"date.time"] <- c("9/30/2016 09:32","10/1/2016 19:20","10/2/2016 02:04")
fail.miss.conc.date
```

```
##   subject_id subject_uid mod_id_event       date.time
## 1       1566    35885929        8.1_1 9/30/2016 09:32
## 2       1566    35885929        8.1_2 10/1/2016 19:20
## 3       1566    35885929        8.1_3 10/2/2016 02:04
```

```r
write.csv(fail.miss.conc.date, file.path(checkDir,"fixMissingConcDate-fent.csv"))
```

After providing the updated file, the same `run_DrugLevel()` function should be re-run. The output now contains an additional message below the first message saying "fixMissingConcDate-fent.csv read with failures replaced". The conc.out data.frame also contains 3 additional rows with the corrected data.

```r
conc.out <- run_DrugLevel(conc.path=file.path(dataDir,"Fentanyl_conc_mod_id.rds"),
    conc.select=c('mod_id','mod_id_visit','samp','fentanyl_calc_conc'),
    conc.rename=c(fentanyl_calc_conc = 'conc.level', samp= 'event'),
    conc.mod.list=list(mod_id_event = expression(paste(mod_id_visit, event, sep = '_'))),
    samp.path=file.path(dataDir,"Fentanyl_samp_mod_id.rds"),
    samp.mod.list=list(mod_id_event = expression(paste(mod_id_visit, samp, sep = '_'))),
    check.path=checkDir,
    failmiss_fn = 'MissingConcDate-',
    multsets_fn = 'multipleSetsConc-',
    faildup_fn = 'DuplicateConc-',
    drugname=drugname,
    LLOQ=LLOQ,
    demo.list=demo.out)
```

```
## ##########################
## 3 rows need review, see file /private/var/folders/06/
##      0qv1dr5508j_tbzqdjfqjf680000gn/T/RtmpxKeQ2k/Rinst31587fb28159/EHR/
##      examples/str_ex2/checks/failMissingConcDate-fent.csv AND create /private/
##      var/folders/06/0qv1dr5508j_tbzqdjfqjf680000gn/T/RtmpxKeQ2k/
##      Rinst31587fb28159/EHR/examples/str_ex2/checks/fixMissingConcDate-fent.csv
## ##########################
## file /private/var/folders/06/0qv1dr5508j_tbzqdjfqjf680000gn/T/RtmpxKeQ2k/
##      Rinst31587fb28159/EHR/examples/str_ex2/checks/fixMissingConcDate-fent.csv
##      read with failures replaced
## subjects with concentration missing from sample file
## [1] mod_id       mod_id_event
## <0 rows> (or 0-length row.names)
## 1 subjects have multiple sets of concentration data
## 16 total unique subjects ids (including multiple visits) currently in the
##      concentration data
## 15 total unique subjects in the concentration data
## ##########################
## 15 rows need review, see file /private/var/folders/06/
##      0qv1dr5508j_tbzqdjfqjf680000gn/T/RtmpxKeQ2k/Rinst31587fb28159/EHR/
##      examples/str_ex2/checks/multipleSetsConc-fent2021-06-05.csv
## ##########################
## 15 total unique subjects ids (after excluding multiple visits) in the
##      concentration data
## 15 total unique subjects in the concentration data
```

- The `run_DrugLevel` function arguments are as follows:
  - `samp.path`: file path where the sampling time data exist
  - `samp.mod.list`: list containing modifications to variables in the sampling time data
  - `conc.path`: file path where the drug concentration data exist
  - `conc.select`: list of variables in the drug concentration data, which are used for processing
  - `conc.rename`: rename variables for the variables in conc.select
  - `conc.mod.list`: list containing modifications to variables in the drug concentration data
  - `check.path`: file path where the generated files for data checking are stored, and the corresponding data files with fixed data exist
  - `failmiss_fn`: filename stub for missing concentration date check file
  - `multsets_fn`: filename stub for multiple sets of concentration data check file
  - `faildup_fn`: filename stub for duplicate concentration check file
  - `drugname`: drug name of interest (e.g., dex, fent)
  - `LLOQ`: lower limit of quantification (LLOQ)
    - e.g., dexmedetomidine 0.005 ng/mL, fentanyl 0.05 ng/mL
  - `demo.list`: file name for processed demographic file that is used to exclude subjects based on exclusion criteria
- The output of the `run_DrugLevel` function is a dataset for processed drug concentration levels (conc.level) matched with date/time (date.time), as well as necessary identification numbers for further merging with the output from other modules.

Pro-Laboratory

The Pro-Laboratory module processes laboratory data that can be merged with data from other modules. The laboratory data can be processed using:

```r
creat.out <- run_Labs(lab.path=file.path(dataDir,"Fentanyl_creat_mod_id.rds"),
    lab.select = c('mod_id','date.time','creat'),
    lab.mod.list = list(date.time = expression(parse_dates(fixDates(paste(date, time))))))

alb.out <- run_Labs(lab.path=file.path(dataDir,"Fentanyl_alb_mod_id.rds"),
    lab.select = c('mod_id','date.time','alb'),
    lab.mod.list = list(date.time = expression(parse_dates(fixDates(paste(date, time))))))

lab.out <- list(creat.out, alb.out)

str(lab.out)
```

```
## List of 2
##  $ :'data.frame':    266 obs. of  3 variables:
##   ..$ mod_id   : int [1:266] 1 1 1 1 1 1 1 1 1 1 ...
##   ..$ date.time: POSIXct[1:266], format: "2017-02-05 04:00:00" "2017-02-06 05:00:00" ...
##   ..$ creat    : num [1:266] 0.52 0.53 0.42 0.35 0.29 0.28 0.34 0.59 0.54 0.26 ...
##  $ :'data.frame':    44 obs. of  3 variables:
##   ..$ mod_id   : int [1:44] 8 8 8 8 4 6 6 9 10 10 ...
##   ..$ date.time: POSIXct[1:44], format: "2020-07-30 05:23:00" "2020-07-28 03:12:00" ...
##   ..$ alb      : num [1:44] 2.9 2 2.7 4.1 4.1 2.3 2.6 3 3.1 4.2 ...
```

- The `run_Labs` function arguments are as follows:
  - `lab.path`: file path where the laboratory data exist
  - `lab.select`: list of variables in the laboratory data to be retained
  - `lab.mod.list`: list containing modifications to variables in the laboratory data

Build-PK-IV

The Build-PK-IV module creates PK data for IV medications. Both dose data from the *Pro-Med-Str1* module and concentration data from the *Pro-DrugLevel* module are required. Demographic data from the *Pro-Demographic* module and laboratory data from the *Pro-Laboratory* module may optionally be included. The module is semi-interactive; it generates several files to check potential data errors, and get feedback from an investigator. If corrected information ('fix' files) are provided, the module should be re-run to incorporate the corrections. The major functions this module performs are:

- Determine whether each IV dose is valid by comparing to concentration data. Doses outside the time frame window defined by concentration data (by default seven days before first concentration through last concentration) are dropped. See `pkdata::trimDoses()` for more information.
- Resolve duplicate doses. A file with duplicate bolus dose records, if any, will be produced using the `faildupbol_fn` stub appended with drugname. A 'fix' file can be added to correct these records.
  - e.g., if `faildupbol_fn` is 'DuplicateBolus-', the file 'failDuplicateBolus-fent.csv' will be created in the `check.path` directory. The corrected file should replace 'fail' with 'fix' (e.g., 'fixDuplicateBolus-fent.csv'), and include an additional variable called 'flag' with the value 'keep' for the records to be retained.
- Add zero dose values after a gap.
- If demographic data is provided, update the 'maxint' value.
- Combine dose and concentration data into PK data format.
- If laboratory data is provided, merge onto PK data based on ID and time.
- If demographic data is provided, merge onto PK data. Demographic weight is used to impute records with missing dose weight. In addition, subjects who meet the exclusion criteria and those with no demographic data are removed. A file showing the missingness frequency and percent for each variable is produced using the `missdemo_fn` stub appended wih drugname.
  - e.g., if `missdemo_fn` is '-missing-demo', the file 'dex-missing-demo.csv' will be created in the `check.path` directory.
- The final PK data includes ID and standard NONMEM formatted variables:
- time - time of dosing or concentration event
- conc - observed concentration (NA for dosing records)
- amt - dose amount administered (NA for concentration records)
- rate - rate of drug administration (e.g., rate=0 for bolus doses)
- mdv - missing dependent variable (dv) indicator (e.g., 0 = not missing dv, 1 = missing dv)
- evid - event ID (e.g., 0 = observation, 1 = dose event)

If demographic data is provided, the demographic variables named in `demo.vars` will also be included and renamed according to `demo.abbr`.

PK data with IV dosing can be built by the `run_Build_PK_IV` function using:

```
pk_dat <- run_Build_PK_IV(conc=conc.out,
    dose=ivdose.out,
    demo.list=demo.out,
    demo.vars=c('weight', 'weight_demo', 'height', 'gender',
                'ageatsurgery', 'stat_sts', 'cpb_sts',
                'length_of_icu_stay'),
    demo.abbr=c('wgt', 'wgt_demo', 'height', 'gender',
                'age', 'stat', 'cpb', 'loi'),
    lab.dat = lab.out,
    lab.vars = c('creat','alb'),
    pk.vars=c('mod_id_visit', 'time', 'conc', 'dose', 'rate', 'event',
              'other', 'multiple.record', 'date', 'mod_id'),
    drugname=drugname,
    check.path=checkDir,
    missdemo_fn='-missing-demo',
    faildupbol_fn='DuplicateBolus-',
    date.format="%m/%d/%y %H:%M:%S",
    date.tz="America/Chicago")
```

```
## 0 duplicated rows
## The dimension of the PK data before merging with demographics: 234 x 9
## The number of subjects in the PK data before merging with demographics: 15
## The number of subjects in the demographic file, who meet the exclusion
##      criteria: 2
## check NA frequency in demographics, see file /private/var/folders/06/
##      0qv1dr5508j_tbzqdjfqjf680000gn/T/RtmpxKeQ2k/Rinst31587fb28159/EHR/
##      examples/str_ex2/checks/fent-missing-demo.csv
## List of IDs missing at least 1 cpb_sts: 12.1
## The list of final demographic variables: weight
## weight_demo
## height
## gender
## ageatsurgery
## stat_sts
## cpb_sts
## length_of_icu_stay
## Checked: there are no missing creat
## List of IDs missing at least 1 alb: 1.2
## 11.1
## 15.1
## 2.1
## 3.1
## 4.1
## 5.1
## 7.1
## 8.1
## The dimension of the final PK data exported with the key demographics:
##      197 x 17 with 13 distinct subjects (mod_id_visit)
```

The function `pullRealId()` appends the original IDs – `subject_id` and `subject_uid` to the data. The parameter `remove.mod.id=TRUE` can be used to also remove any module IDs – `mod_id`, `mod_visit`, and `mod_id_visit`.

```r
# convert id back to original IDs
pk_dat <- pullRealId(pk_dat, remove.mod.id=TRUE)

head(pk_dat)
```

```
##      subject_id subject_uid time      conc   amt rate mdv evid   wgt wgt_demo
## 2         466.1    28579217 0.00        NA  50.0  0.0   1    1 21.99    21.99
## 2.1       466.1    28579217 0.25        NA  20.0  0.0   1    1 21.99    21.99
## 2.2       466.1    28579217 4.25 0.2798208    NA   NA   0    0 21.99    21.99
## 12       1607.0    38551767 0.00        NA 109.2 10.4   1    1  2.60     2.76
## 12.1     1607.0    38551767 0.00        NA  10.0  0.0   1    1  2.60     2.76
## 12.2     1607.0    38551767 1.25        NA  15.0  0.0   1    1  2.60     2.76
##      height gender  age stat cpb loi creat alb
## 2    116.90      0 2451    1 107   1  0.54  NA
## 2.1  116.90      0 2451    1 107   1  0.54  NA
## 2.2  116.90      0 2451    1 107   1  0.54  NA
## 12    45.94      0   23    3 110  12  0.66 1.6
## 12.1  45.94      0   23    3 110  12  0.66 1.6
## 12.2  45.94      0   23    3 110  12  0.66 1.6
```

- The `run_Build_PK_IV` function arguments are as follows:
  - `conc`: concentration data output from run_DrugLevel (required)
  - `dose`: IV dose data output from run_MedStrI (required)
  - `demo.list`: (optional) file name for processed demographic file
  - `demo.vars`: (optional) vector of demographic variables to include in final output
  - `demo.abbr`: (optional) vector of abbreviations for demographic variables in `demo.vars`
  - `lab.dat`: (optional) list containing processed laboratory files
  - `lab.vars`: (optional) vector containing names for laboratory files
  - `pk.vars`: PK variables to include
  - `drugname`: drug name stub (e.g., fent)
  - `check.path`: file path where the generated files for data checking are stored, and the corresponding data files with fixed data exist
  - `missdemo_fn`: filename stub for report of missingness frequency and percent for variables
  - `faildupbol_fn`: filename stub for duplicate bolus dose records
  - `date.format`: date and time format (e.g., "%m/%d/%y %H:%M:%S")
  - `date.tz`: date timezone (e.g., "America/Chicago")
- The output of the `run_MedStrI` function is a dataset with NONMEM formatted PK variables and (optionally) demographic and laboratory data.

References

1. Choi L, Beck C, McNeer E, Weeks HL, Williams ML, James NT, Niu X, Abou-Khalil BW, Birdwell KA, Roden DM, Stein CM. Development of a System for Post-marketing Population Pharmacokinetic and Pharmacodynamic Studies using Real-World Data from Electronic Health Records. Clinical Pharmacology & Therapeutics. 2020 Apr; 107(4): 934-943.