Assembling Pharmacometric Datasets in R - The puzzle Package

by Mario González-Sales, Olivier Barrière*, Pierre Olivier Tremblay, Guillaume Bonnefois, Julie Desrochers and Fahima Nekka*

Highlights

- Pharmacometric analyses are integral components of the drug development process.
- The time required to construct a pharmacometrics dataset can sometimes be higher than the effort required for the modeling *per se*.
- The puzzle package is the first open source tool developed to simplify and facilitate the time consuming and error prone task of assembling pharmacometrics datasets.
- The datasets created using the puzzle package are compatible with the format requirements for the gold-standard non-linear mixed effects modeling software.
- The puzzle package is available on CRAN.

Abstract

Background and objective: Pharmacometric analyses are integral components of the drug development process. The core of each pharmacometric analysis is a dataset. The time required to construct a pharmacometrics dataset can sometimes be higher than the effort required for the modeling *per se*. To simplify the process, the puzzle R package has been developed aimed at simplifying and facilitating the time consuming and error prone task of assembling pharmacometrics datasets.

Methods: Puzzle consist of a series of functions written in R. These functions create, from tabulated files, datasets that are compatible with the formatting requirements of the gold standard non-linear mixed effects modeling software.

Results: With only one function, `puzzle()`, complex pharmacometrics databases can easily be assembled. Users are able to select from different absorption processes such as zero- and first-order, or a combination of both. Furthermore, datasets containing data from one or more analytes, and/or one or more responses, and/or time dependent and/or independent covariates, and/or urine data can be simultaneously assembled.

Conclusions: The puzzle package is a powerful and efficient tool that helps modelers, programmers and pharmacometricians through the challenging process of assembling pharmacometrics datasets.

Introduction

The pharmacometrics workflow has routine steps: 1) assemble the dataset, 2) explore, 3) model the data, 4) evaluate, 5) validate the model, and 6) communicate the findings. The automation of these steps saves time and money, reduces the risk of errors, and increases reproducibility. Currently, a number of excellent tools are available to enhance steps 2-6 (Jonsson and Karlsson, 1999; Lindbom et al., 2005; Keizer et al., 2011; Keizer, 2015; Mouksassi, 2016; Wang et al., 2016; Keizer, 2018; Xie et al., 2018; Mouksassi, 2019; Baron, 2019; RStudio team). However, to the best of our knowledge, there is no open source tool to support step 1). Considerable challenges exist when working in data management with an industrial setting that must comply with rules and regulations under the scrutiny and control of regulatory agencies, such as the U.S. Food and Drug Administration (FDA), Health Canada, the Japanese Pharmaceutical and Medical Device Agency (PMDA), or the European Medicines Agency (EMA). The task of dataset building should not be underrated, since the amount of time required to construct a pharmacometrics dataset can be sometimes higher than the effort required for the modeling *per se*. In fact, it is often claimed that the process of cleaning and preparing the data could take up to 80% of data analysis (Dasu and Johnson, 2003). In addition, data preparation does not amount to a single step, since it usually has to be repeated over the course of analysis as new data become available.

Understanding the structure of pharmacometrics datasets is essential for an efficient data assembling. These datasets are two-dimensional arrangements of data in rows and columns. Each row represents a record or an event, while each column represents an item or a variable. Pharmacokinetic (PK) datasets are generally time-ordered arrangements of records representing the time course of plasma concentrations related to the dose of drug administered. Depending on the complexity of the system being modeled, pharmacodynamics (PD) datasets might or not include dose and/or time information. Furthermore, pharmacometrics datasets normally imply multiple doses, several routes of administration, different treated arms and/or sequences, time dependent and/or independent covariates, metabolite, and/or urine data, and/or information regarding one or multiple PD endpoints. These additional components tend to skyrocket the complexity of the data assembling as it makes the process hard to script up-front. This is especially true when working with data that are not in tidy shape (Wickham et al., 2014). Due to this entanglement, complications are likely to happen throughout the process. In this regard, inconsistencies between exact dates, times of dose and blood sampling, different identifiers on the same requisition forms, imputation of missing and time-varying covariates, and missing timing of concomitant medications are common issues (Grasela et al., 2007).

In this tutorial, we present the **puzzle** package to the pharmacometrics community, an open source tool for assembling pharmacometrics datasets in R. The **puzzle** package consists in a group of functions developed to simplify and facilitate the time consuming and error prone task of assembling pharmacometrics datasets. The datasets created are compatible with the formatting requirements of the NONMEM® software (Beal et al., 2009). Moreover, as NONMEM® data structures are the gold standard for non-linear mixed effects modeling, the datasets created with the puzzle package are mostly compatible with other non-linear mixed effects softwares such as MONOLIX (Lixoft), **saemix** (Comets et al., 2011), and **nlmixr** (Fidler et al., 2019). The **puzzle** package is also available on CRAN and can be installed with the following R code: `install.packages("puzzle")`.

This tutorial proceeds as follows. First, we illustrate two common case studies of data assembling of pharmacometrics datasets using the **puzzle** package. Second, we dive into the arguments of the main function of the **puzzle** package, the `puzzle()` function. Third, we present the pre-formatting requirements of `puzzle()`, and finally, we conclude with a discussion regarding the limitations and inconveniences of this framework, and what are the other approaches or efforts that might be advantageous to pursue.

Use of the puzzle function

The `puzzle()` function is flexible enough to build all types of pharmacometrics datasets in the appropriate format for the pharmacometrics analysis. The data presented in Figure 1 will be used to show the readers the flexibility of the `puzzle()` function and how powerful it is. In the first example, panels b, d, and e of Figure 1 will be used to assemble a PK dataset containing parent and metabolite plasma concentrations, several covariates, and multiple administrations. The second example will imply panels a, c, d and f of Figure 1. The output dataset will be a PK/PD data set with time dependent covariates.

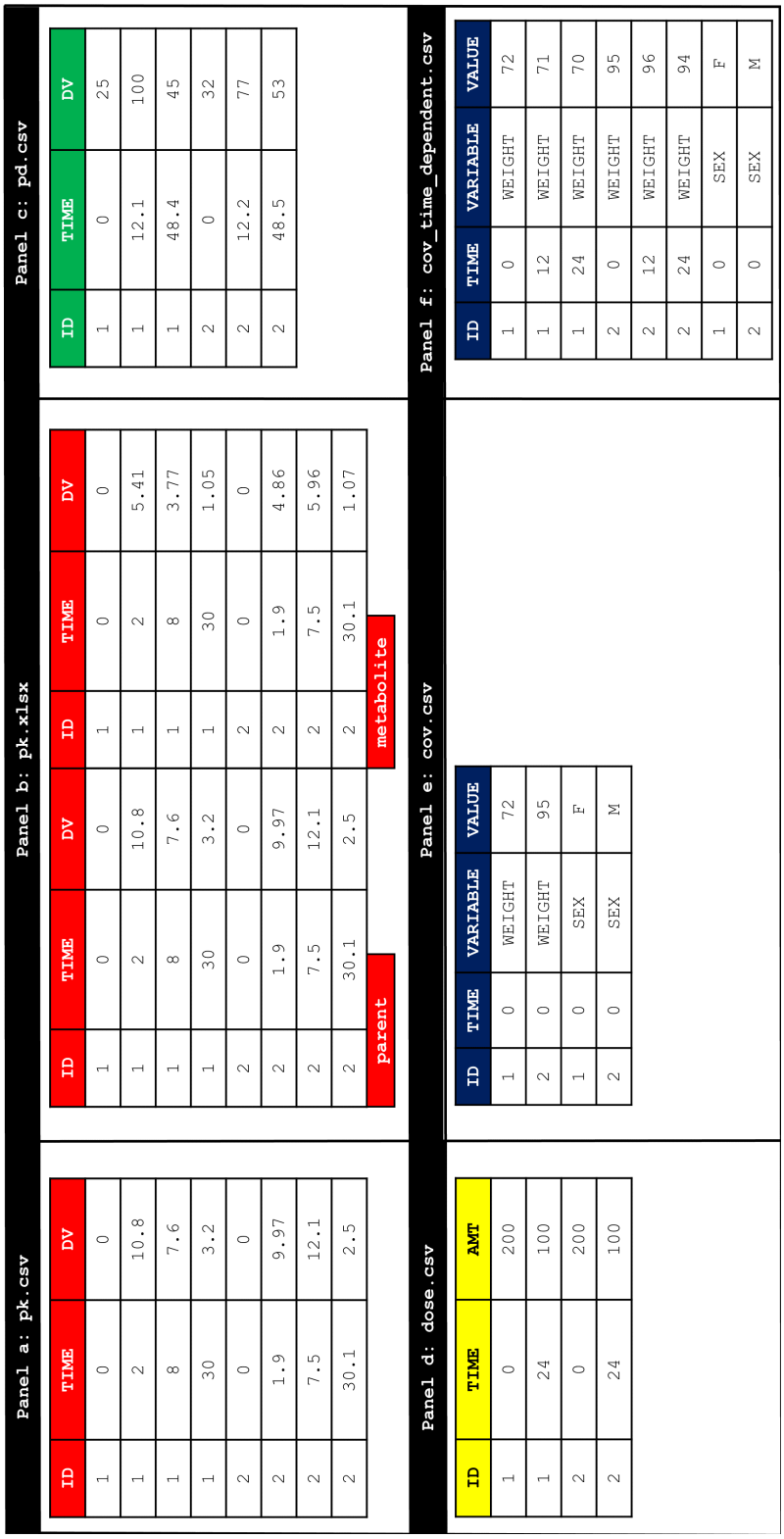

Figure 1: Structure of the four different files to input data into `puzzle()` function. *Panels a and b illustrate the items required to input PK information from one and two analytes, respectively. Panel c and d depict the items to input PD and dosing information. Panel e and f presents the pre-formatting requirements to input covariate information assuming time in- and dependent covariates, respectively.*

Example 1: PK data set

The syntax to create the NM_PK.csv depicted in Table 1 is as follows:

```
library(puzzle)
directory = "C:/projects/puzzle/data"
puzzle(directory=file.path(directory),
       order=c(1,1),
       pk=list(name="pk.xlsx"),
       dose=list(name="dose.csv"),
       cov=list(name="cov.csv"),
       nm=list(name="NM_PK.csv"),
       username="Mario Gonzales Sales")
```

C	ID	TIME	TAD	DOSETIME	PDOSETIME	NUMDOSE	AMT	CMT	EVID	DV	LDV	MDV	SEX	WEIGHT
	1	0	0	0	0	1	200	1	1	.	.	1	0	72
	1	0	0	0	0	1	200	2	1	.	.	1	0	72
	1	0	0	0	0	1	.	3	0	0	.	0	0	72
	1	0	0	0	0	1	.	4	0	0	.	0	0	72
	1	2	2	0	0	1	.	3	0	10.8	2.37954613	0	0	72
	1	2	2	0	0	1	.	4	0	5.41	1.68824909	0	0	72
	1	8	8	0	0	1	.	3	0	7.6	2.02814825	0	0	72
	1	8	8	0	0	1	.	4	0	3.77	1.327075	0	0	72
	1	24	0	24	0	2	100	1	1	.	.	1	0	72
	1	24	0	24	24	2	100	2	1	.	.	1	0	72
	1	30	6	24	24	2	.	3	0	3.2	1.16315081	0	0	72
	1	30	6	24	24	2	.	4	0	1.05	0.04879016	0	0	72
	2	0	0	0	0	1	200	1	1	.	.	1	1	95
	2	0	0	0	0	1	200	2	1	.	.	1	1	95
	2	0	0	0	0	1	.	3	0	0	.	0	1	95
	2	0	0	0	0	1	.	4	0	0	.	0	1	95
	2	1.9	1.9	0	0	1	.	3	0	9.97	2.29958058	0	1	95
	2	1.9	1.9	0	0	1	.	4	0	4.86	1.58103844	0	1	95
	2	7.5	7.5	0	0	1	.	3	0	12.1	2.49320545	0	1	95
	2	7.5	7.5	0	0	1	.	4	0	5.96	1.78507048	0	1	95
	2	24	0	24	0	2	100	1	1	.	.	1	1	95
	2	24	0	24	24	2	100	2	1	.	.	1	1	95
	2	30.1	6.1	24	24	2	.	3	0	2.5	0.91629073	0	1	95
	2	30.1	6.1	24	24	2	.	4	0	1.07	0.06765865	0	1	95

Table 1: Returned output from the `puzzle()` function after running example 1

After running the `puzzle()` function the following message will be printed in the R console:

```
Automatic coercion to numeric for CMT
3=parent
4=metabolite
Automatic coercion to numeric for SEX
0=F
1=M
Assembling date and time: 2019-10-29 12:12:47
Time zone: Europe/Paris
Number of individuals: 2
Number of observations: 16
Dose levels: “100”, “200”
This data set was assembled by Mario Gonzalez Sales
```

It informs the user that parent and metabolite records are located in CMT 3 and 4, respectively. Moreover, it also indicates that the categorical variable SEX has been coerced into a numeric variable. Female (F) and male (M) categories have been assigned to the values of 0 and 1, respectively. The user should be aware that factors are coerced following alphabetical order. The date and time at which the dataset was assembled, the timezone where the assembling was performed, the number of individuals and observations in the dataset, the different dose levels administered, and if requested, the person who performed the data assembling will be automatically displayed.

The `puzzle()` function simultaneously assembles the PK (*i.e.* parent and metabolite), the dose and the covariate information returning a NONMEM® ready dataset stored in a .csv file (*i.e.* "NM_PK.csv").

Because order has been set to `order = c(1,1)`, the dose has been split into two compartments (*i.e.* CMT 1 and 2). Furthermore, because there are two analytes, the observations have been assigned to an independent compartment (*i.e.* CMT 3 and 4) for the parent and the metabolite, respectively. Additionally, the `puzzle()` function automatically appends useful items to the output. Specifically, a placeholder to ignore records (C), time after dose (TAD), dosing time (DOSETIME), prior dosing time (PDOSETIME), the compartment item (CMT), the event identification (EVID), the log of the DV (LDV) and the missing dependent variable item (MDV).

Example 2: PK/PD data set

The syntax to create the NM_PKPD.csv depicted in Table 2 is as follows:

```
puzzle(directory=file.path(directory),
       order=0,
       pk=list(name="pk.csv"),
       pd=list(name="pd.csv"),
       dose=list(name="dose.csv"),
       cov=list(name="cov_time_dependent.csv"),
       coercion=list(name="coercion.txt"),
       nm=list(name="NM_PKPD.csv"))
```

C	ID	TIME	TAD	DOSETIME	PDOSETIME	NUMDOSE	AMT	TYPE	CMT	EVID	DV	LDV	MDV	SEX	WEIGHT
	1	0	0	0	0	1	200	0	1	1	.	.	1	0	72
	1	0	0	0	0	1	.	1	1	0	0	.	0	0	72
	1	0	0	0	0	1	.	2	2	0	25	3.21887582	0	0	72
	1	2	2	0	0	1	.	1	1	0	10.8	2.37954613	0	0	72
	1	8	8	0	0	1	.	1	1	0	7.6	2.02814825	0	0	72
	1	12.1	12.1	0	0	1	.	2	2	0	100	4.60517019	0	0	71
	1	24	0	24	0	2	100	0	1	1	.	.	1	0	70
	1	30	6	24	24	2	.	1	1	0	3.2	1.16315081	0	0	70
	1	48.4	24.4	24	24	2	.	2	2	0	45	3.80666249	0	0	70
	2	0	0	0	0	1	200	0	1	1	.	.	1	1	95
	2	0	0	0	0	1	.	1	1	0	0	.	0	1	95
	2	0	0	0	0	1	.	2	2	0	32	3.4657359	0	1	95
	2	1.9	1.9	0	0	1	.	1	1	0	9.97	2.29958058	0	1	95
	2	7.5	7.5	0	0	1	.	1	1	0	12.1	2.49320545	0	1	95
	2	12.2	12.2	0	0	1	.	2	2	0	77	4.34380542	0	1	96
	2	24	0	24	0	2	100	0	1	1	.	.	1	1	94
	2	30.1	6.1	24	24	2	.	1	1	0	2.5	0.91629073	0	1	94
	2	48.5	24.5	24	24	2	.	2	2	0	53	3.97029191	0	1	94

Table 2: Returned output from the `puzzle()` function after running example 2

In this case, the variable type of observation (*i.e.* TYPE) has been appended to the "NM_PKPD.csv" file. Because a bolus administration has been specified with the argument '`order`' (*i.e.* `order = 0`), TYPE has a value of 0 for dosing events, and a value of 1 and 2 for PK and PD records, respectively. Moreover, the file "coercion.txt" has been generated. The content is printed below:

```
CMT: 1=pk, 2=pd
SEX: 0=F, 1=M
```

The argument of the puzzle function

The `puzzle()` function builds pharmacometrics ready datasets from a group of tabulated files. For convenience, it always returns a ".csv" file or an R object of class "data.frame" as output. The path to the location of the files to be assembled can be set with the '`directory`' argument. Depending on the pharmacometric analysis to be performed, the tabulated files may contain PK, dose, covariates and/or PD information.

The PK data may include drug concentration (*i.e.* parent and/or metabolite/s) and/or urine information. The '`pk`' argument is used to input this type of data into the `puzzle()` function. The general form is as follows:

```
pk = list(name = NULL, data = NULL)
```

If the PK data is stored in a ".csv" or an ".xlsx" file, the name of the file should be provided with name:

```
pk = list(name = "pk.csv", data = NULL)
```

Alternatively, if the data is within the R environment, the name of the R object may be defined with the parameter data:

```
pk = list(name = NULL, data = nm$pk)
```

where 'nm' is a list containing the PK information 'pk'. Excel files are useful to simultaneously store PK information from two or more analytes. The `puzzle()` function is smart enough to assemble all the PK information at once. To this end, each analyte data has to be contained in its own spreadsheet. Moreover, each sheet has to have the same items, and the items have to be in the same order.

The PD information may comprise the time course of the clinical endpoint of interest, time to event data, and/or a response outcome. This type of information is handled with the 'pd' argument. Its behavior is similar to 'pk', and thus, it accepts the same type of tabulated files and it requires a similar syntax. For example, in case of multiple endpoints stored in an ".xlsx" file, the user should define:

```
pd = list(name = "pd.xlsx", data = NULL)
```

or simply

```
pd = list(name = "pd.xlsx")
```

Consistently, covariate and dose information are inputted to the `puzzle()` function with the arguments 'cov' and 'dose', respectively. The reader should note that ".xlsx" files are not required, and the data should be contained within a ".csv" file or an R object. Finally, 'extratimes' is another argument following this logic. This argument can be used to add a common pattern of additional times for each individual within the dataset. The user may be interested in this feature for prediction purposes. For example, if the data are very sparse (*i.e.* 2 observations per individual from time 1 to 4 hours post drug administration), it is possible to define a vector of times from time 0 to 4 hours post dose in order to obtain a smooth prediction (*e.g.* `extratimes = list(name = "extratimes.csv",data = NULL)`). Of note, EVID = 2 will be assigned to these extra times.

The 'cov' and 'dose' arguments may be used in combination with a number of additional arguments: 'optionalcolumns', which defines the optional columns to be appended in to the output. For instance, if more than one treatment is administered, the file storing the dose information may contain the variable treatment (*i.e.* "TRT"). If the user wants to include this variable in the pharmacometrics dataset, the following syntax is warranted: `optionalcolumns = "TRT"`. If more than one variable are to be appended, the syntax should be: `optionalcolumns = c("variable 1","variable 2")`. Furthermore, the argument 'fillcolumns' fills the columns appended with the argument 'optionalcolumns' using the last observation carried forward method. If the argument is defined as `fillcolumns = "TRT"`, the variable TRT will be added without missing values. Otherwise, the value of TRT only will be available for dosing records, and non-dosing records will be outputted as ".", the default for missing values.

The `puzzle()` function is able to coerce numeric variables from character variables. If a character variable is introduced with the 'cov' argument, this variable will be automatically coerced into a numeric variable. For example, let's assume we have a character variable accounting for renal impairment with the following possible values: "mild", "moderate" and "severe". The `puzzle()` function will convert the variable from character to factor. Each string will be a different level. Then, the factor variable will be coerced into numeric. If there is no numeric order within the factor, the levels will be alphabetically assigned. The argument 'initialindex' defines the initial value of the coerced numeric variable. The default value is `initialindex = 0`. Therefore, under the default settings, the "mild", "moderate" and "severe" levels will have numeric values of 0, 1 and 2, respectively. Moreover, a string for missing values can be defined with the argument 'na.strings'. Thus, if a variable has a missing value, this can be labeled as not available as follows: `na.strings = "N/A"`. For convenience, a ".txt" file may be created with the argument 'coercion'. This argument creates a record of the categories of each coerced variable. If the user defines `coercion = list(name = "my_coercion_records.txt")`, a file with the name "my_coercion_records.txt" will be generated in the working directory after running the `puzzle()` function. The user can also control the variables to be included in the records with the argument 'nocoercioncolumns'.

When performing population PK analysis in NONMEM®, the structure of the data set depends on the route of administration. This implies that the data structure is different for bolus, infusion and oral administrations (Owen and Fiedler-Kelly, 2014). The argument 'order' is used to handle this situation. It can take four different values: `order = 0` indicates a zero-order absorption process. It may be used for bolus or infusion administrations. For the later, the RATE has to be given in the dosing file; `order = 1` serves to generate datasets to model first-order absorption processes (*e.g.* oral administration); `order = c(1,1)` may be used to model the absorption of drugs with complex formulations. An example may be a drug formulated as immediate and extended release. In this

situation, two different absorption rates may be warranted, and `c(1,1)` means that the dataset is built assuming two first-order absorption processes. Finally, `order = c(0,1)` allows the user to define a zero- and first-order absorption processes. By default, the `puzzle()` function assumes this process occurs in parallel, meaning that one fraction of the drug is absorbed thorough a zero-order process and the remaining amount of the drug is absorbed following a first-order process. However, it may also follow a sequential process where the drug is delivered thorough a zero-order process to the depot compartment, and then, absorbed into the bloodstream following a first-absorption process. The argument '`parallel`' can be set to false (*i.e.* `parallel = FALSE`) to control these absorption patterns. The user must be aware that if order is not set to `order = c(0,1)` and `parallel = FALSE`, `puzzle()` will return the following error message:

Error in puzzle(directory = file.path(getwd()), order = c(1, 1), parallel = F,: Would you like to use a sequential zero + first order absorption model? Please set order=c(0,1). Otherwise, please set parallel = T

A characteristic of the NONMEM® datasets is that characters are not allowed, and all the items, except DAT, must be numeric. Hence, the command IGNORE in the $INPUT block within a NONMEM® control stream is commonly used to ignore the label of the data items. The `puzzle()` function has the argument '`ignore`' to create a new item for this purpose. For example, if the user sets ignore = "C", the first column of the returned ".csv" file will contain an item called C. The argument '`missingvalues`' allows the user to specify a label for the missing values. The default value is `missingvalues = "."`.

The user can control how the records are arranged with the argument '`arrange`'. By default, records are arranged as follows: `arrange="ID,TIME,CMT,desc(EVID)"`. Moreover, complex date formatting is also supported. The argument '`datetimeformat`' defines the format for dates and times. By default, the format is `datetimeformat = "%Y-%m-%d %H:%M:%S"`. In addition to that, time units may be specified with the argument '`timeunits`'. For example, `timeunits = "hours"`. It should be highlighted that the timezone will affect how times are computed from dates. Thus, depending on your location, times may be slightly different because the default value is `timezone = Sys.timezone()`. The user can also identify the person assembling the dataset with the argument `username`. The last argument of the `puzzle()` function is '`nm`'. It is used to name the output from the function, in other words, the returned ".csv" file. The syntax of the arguments of the `puzzle()` function is presented in Table 3.

Argument	Example of syntax
directory	`directory = file.path(getwd())`
pk	`pk = list(name="pk.csv")`
pd	`pd = list(name="pd.csv")`
cov	`cov = list(name="cov.csv")`
dose	`dose = list(name="dose.csv")`
extratimes	`extratimes = list(name="extratimes.csv")`
optionalcolumns	`optionalcolumns = "TRT"`
fillcolumns	`fillcolumns = "TRT"`
initialindex	`initialindex = 0`
na.strings	`na.strings = "N/A"`
coercion	`coercion = list(name = "coercion.txt")`
nocoercioncolumns	`nocoercioncolumns = NULL`
order	`order = c(0,1)`
parallel	`parallel = FALSE`
ignore	`ignore = "C"`
missingvalues	`missingvalues = "."`
arrange	`arrange = "ID,TIME,CMT,desc(EVID)"`
datetimeformat	`datetimeformat="%Y-%m-%d %H:%M:%S"`
timeunits	`timeunits="hours"`
timezone	`timezone=Sys.timezone()`
username	`Username="User"`
nm	`nm=list(name="NONMEM.csv")`

Table 3: Syntax of the arguments of the `puzzle()` function

Pre-formatting requirements

At the beginning of a new project, the modeler or the programmer usually faces a series of large and structurally diverse datasets where the required information for the analysis is stored. It is a common practice that data are collected using automatic informatics methods. Unfortunately, the systems may storage the information in multiple and non-tidy pre-defined files. This situation is not ideal to assemble the pharmacometrics datasets because the programmer has to spend valuable time puzzling out and assembling all the heterogeneous information. The `puzzle()` function generates NONMEM® formatted datasets from standard tabulated files. Consequently, before calling the `puzzle()` function, it may be necessary to perform some data manipulation (Wickham and Grolemund, 2016; Chen et al., 2017). The degree of data manipulation will depend on the structure of the stored information. Given the high combination of possible data structures, detailing how the data should be manipulated is out of the scope of this manuscript. The reader is referred to the documentation of the **tidyverse** package for more information regarding how to efficiently perform this step (Wickham, 2017).

Pharmacometrics datasets usually contain PK, PD, dose and/or covariate information. The `puzzle()` function requires each type of information to be inputted from a separate tabulated file. Each file has to contain pre-defined variables with specific labels.

Pre-formatting requirements for the tabulated file containing the PK information

This file has to have at least three columns: i) a column used for subject identification "ID"; ii) a column containing the sampling times at which the PK samples were collected "TIME" or "DATETIME"; and iii) the observed value of the dependent variable "DV". The tabulated file can be an R object of class list, or a ".csv" file in case of assembling data from one analyte (Figure 1: panel a). If information from more than one analyte is to be assembled (Figure 1: panel b), the data should be stored in an ".xlsx", or in an R object of class list. For example, if parent and metabolite information are available, the `puzzle()` function requires an ".xlsx" file with one sheet containing the PK information of the parent drug, and another sheet storing the PK information of the metabolite. There is no limit in the number of sheets that the `puzzle()` function can handle. However, it is mandatory that each sheet includes the three pre-defined columns mentioned above.

Pre-formatting requirements for the tabulated file containing the PD information

The `puzzle()` function has the same requirements for the PD than for the PK information. Therefore, the same three items (*i.e.* ID, TIME or DATETIME, and DV) are mandatory and depending on the number of PD endpoints available, the data can be stored in an R object of class list, a ".csv" or an ".xlsx" file (Figure 1: panel c).

Pre-formatting requirements for the tabulated file containing the dosing records

At least three columns are required: i) a column used as subject identification "ID": ii) a column containing the times at which the doses were administered "TIME" or "DATETIME"; and iii) the given dose "AMT" (Figure 1: panel d). In addition to that, and depending on the study design, other columns may be present as well. Later on, these columns may be passed to the returned ".csv" file using the '`optionalcolumns`' argument.

Pre-formatting requirements for the tabulated file containing the covariate information

If this file is used, it has to include the following three items: i) a column with subject identification "ID"; ii) the name of the variable "VAR", and iii) the value of the variable "VALUE" (Figure 1: panel e). Of note, the `puzzle()` function is able to handle time independent (*e.g.* sex) or time dependent covariates (Figure 1: panel f). In the specific case of time dependent covariates, a fourth item is mandatory: iv) a column containing the times at which the covariates were collected "TIME" or "DATETIME". This is required so that the `puzzle()` function can know the times at which the value of the covariate changes within a subject.

Discussion

The **puzzle** package has been designed to be used as simple as possible. In fact, with a few lines of R code, modelers, programmers and overall pharmacometricians have now an open source tool to assemble complex pharmacometrics datasets. In order to facilitate its use and to decrease the slope of the learning curve, users are only required to learn the behavior and syntax of one function, `puzzle()`. Nevertheless, the **puzzle** package involves additional functions intentionally working "under the hood" to enhance the user experience. These functions are borrowed from several R libraries including: utils, lubridate, stats, readxl, reshape2, sqldf, plyr, and dplyr. The **puzzle** package started to be coded more than six years ago. At the time, reshape2, plyr, readxl and sqldf were useful tools from a data assembling perspective. However, if the **puzzle** package was started to be coded from scratch nowadays, the authors would probably use the **tidyverse** package as reference.

At first, the main inconvenience of `puzzle()` may be the requirement of specific pre-defined formatting for its inputs. Nevertheless, it is indeed this feature that drastically increases the flexibility and productivity of `puzzle()`, allowing the assembling of all types of pharmacometrics datasets. The examples provided herein only scratches the surface of what the package is able to do. The reader is referred to the supplementary material for additional examples on how assembling pharmacometrics datasets using the **puzzle** package.

The data assembling workflow starts with data collection. The second step is data storage. Unfortunately, most of the heterogeneity and difficulty during the data assembling arises in this step. Effectively, each company or hospital may use its own recipe to store the data. Even more complexity is usually added to the process if, within a company or hospital, different scientists or nurses are involved in the data collection and/or storage. This is due to inter-individual (*i.e.* "two persons may label the same item differently") and/or inter-occasion (*i.e.* "the same person may label the same item differently") variability. This fact makes the principle of reproducibility from a data preparation point of view quite challenging.

The use of Study Data Tabulation Model (SDTM) has provided a standard for organizing and formatting data to streamline processes in collection, management, analysis and reporting, and it is one of the required standards for data submission to FDA and PMDA. The submission data standards team of Clinical Data Interchange Standards Consortium (CDISC) defines SDTM (Clinical Data Interchange Standards Consortium documentation). On July 21, 2004, SDTM was selected as the standard specification for submitting tabulation data to the FDA for clinical trials and on July 5, 2011 for nonclinical studies.

Unfortunately, inconsistencies in the data are not uncommon even if strict rules are supposed to be followed. Because of the heterogeneity of the data being assembled, the main limitation of the **puzzle** package is the requirement of pre-data manipulation. Specifically, the inconsistencies from SDTM have to be cleared before its use. Thus, at this stage of the development, the **puzzle** package is not fully compatible with SDTM datasets. Nevertheless, it is planned to implement this feature in the next version of the package. Alternatively, maybe, it is time within the pharmacometrics community to go one step further and develop new tools and methods allowing the establishment of strategies and/or protocols where data are more homogeneously stored. If this objective is accomplished and the degree of pre-data manipulation is reduced or even completely eliminated, the often-painful process of data assembling may become a breeze.

Conclusion

The **puzzle** package is a very flexible, powerful and efficient tool that helps modelers, programmers and/or pharmacometricians through the complex model building process. Specifically, it is the first open source tool supporting pharmacometricians during the difficult, error prone, and time consuming process of data assembling. Therefore, the **puzzle** package fills an unmet condition within the pharmacometrics workflow as it offers an opportunity for a unification of the code for data assembly improving reproducibility and traceability. Thus, we do believe that the **puzzle** package will be of high interest for the pharmacometrics community. It is the authors hope that this manuscript serves as detonating to set the foundations for a more efficient and pleasant data assembling in our field.

Bibliography

K. T. Baron. The mrgsolve package. https://mrgsolve.github.io, 2019. Accessed 2019-07-08. [p2]

S. Beal, L. Sheiner, A. Boeckmann, and R. Bauer. Nonmem user's guide. (1989-2009), 2009. Icon Development Solutions, Ellicott City, MD, USA. [p2]

S.-Y. Chen, Q. Liu, and Z. Feng. Common data manipulations with r in biological researches. *Journal of thoracic disease*, 9(7):2209–2213, July 2017. ISSN 2072-1439. doi: 10.21037/jtd.2017.06.48. [p8]

Clinical Data Interchange Standards Consortium documentation. https://www.cdisc.org/standards/foundational/sdtm. Accessed 2019-11-29. [p9]

E. Comets, A. Lavenu, and M. Lavielle. saemix: Stochastic Approximation Expectation Maximization (SAEM) algorithm. https://rdrr.io/cran/saemix/man/saemix.html, 2011. Accessed 2019-11-29. [p2]

T. Dasu and T. Johnson. *Exploratory Data Mining and Data Cleaning*. Wiley-IEEE, 2003. [p2]

M. Fidler, J. J. Wilkins, R. Hooijmaijers, T. M. Post, R. Schoemaker, M. N. Trame, Y. Xiong, and W. Wang. Nonlinear mixed-effects model development and simulation using nlmixr and related r open-source packages. *CPT: pharmacometrics & systems pharmacology*, 8:621–633, Sept. 2019. ISSN 2163-8306. doi: 10.1002/psp4.12445. [p2]

T. H. Grasela, J. Fiedler-Kelly, B. Cirincione, D. Hitchcock, K. Reitz, S. Sardella, and B. Smith. Informatics: the fuel for pharmacometric analysis. *The AAPS journal*, 9(1):E84–E91, Mar. 2007. ISSN 1550-7416. doi: 10.1208/aapsj0901008. [p2]

E. N. Jonsson and M. O. Karlsson. Xpose–an S-PLUS based population pharmacokinetic/pharmacodynamic model building aid for nonmem. *Computer methods and programs in biomedicine*, 58(1):51–64, Jan. 1999. ISSN 0169-2607. [p2]

R. J. Keizer. R library for simulation of PKPD models defined as ODE systems. The PKPDsim package. https://github.com/ronkeizer/PKPDsim, 2015. Accessed 2019-07-08. [p2]

R. J. Keizer. R library to create visual predictive checks. The vpc package. https://github.com/ronkeizer/vpc, 2018. Accessed 2019-07-08. [p2]

R. J. Keizer, M. van Benten, J. H. Beijnen, J. H. M. Schellens, and A. D. R. Huitema. Piraña and pcluster: a modeling environment and cluster infrastructure for nonmem. *Computer methods and programs in biomedicine*, 101(1):72–79, Jan. 2011. ISSN 1872-7565. doi: 10.1016/j.cmpb.2010.04.018. [p2]

L. Lindbom, P. Pihlgren, E. N. Jonsson, and N. Jonsson. Psn-toolkit–a collection of computer intensive statistical methods for non-linear mixed effect modeling using nonmem. *Computer methods and programs in biomedicine*, 79(3):241–257, Sept. 2005. ISSN 0169-2607. doi: 10.1016/j.cmpb.2005.04.005. [p2]

Lixoft. MONOLIX 2019 version documentation. http://monolix.lixoft.com/. Accessed 2019-10-28. [p2]

S. Mouksassi. Rshiny app as interface to ggplot2. The ggplotwithyourdata package. https://github.com/smouksassi/ggplotwithyourdata, 2016. Accessed 2019-07-08. [p2]

S. Mouksassi. Ggplot and summary statistics quick exploration of data. The ggquickeda package. https://github.com/smouksassi/ggquickeda, 2019. Accessed: 2019-07-08. [p2]

J. Owen and J. Fiedler-Kelly. *Introduction to population pharmacokinetic and pharmacodynamic analysis with nonlinear mixed effects models.* Wiley, 2014. [p6]

RStudio team. Rmarkdown: dynamic documents for R. The rmardown package. https://rmarkdown.rstudio.com/. Accessed 2019-10-28. [p2]

W. Wang, K. M. Hallow, and D. A. James. A Tutorial on RxODE: Simulating Differential Equation Pharmacometric Models in R. *CPT: pharmacometrics & systems pharmacology*, 5(1):3–10, Jan. 2016. ISSN 2163-8306. doi: 10.1002/psp4.12052. [p2]

H. Wickham. The tidyverse package. https://www.tidyverse.org/, Nov. 2017. Accessed 2019-07-08. [p8]

H. Wickham and G. Grolemund. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data.* O'Really Media, 2016. [p8]

H. Wickham et al. Tidy data. *Journal of Statistical Software*, 59(10):1–23, 2014. [p2]

Y. Xie, J. J. Allaire, and G. Grolemund. *R Markdown: The Definitive Guide*. Chapman & Hall/CRC, 2018. [p2]

*Mario González Sales**
Modeling Great Solutions
Crta. Engolasters s/n, Escaldes-Engordany, AD700
Andorra
mario@modelinggreatsolutions.com

*Olivier Barrière**
Certera
2000 Peel Street, Suite 570, Montréal, H3A 2W5
Canada
olivier.barriere@certara.com

*Both authors share the first authorship.

Pierre Olivier Tremblay
Syneos Health
2500 Rue Einstein, Québec, G1P 0A2
Canada
pierreolivier.tremblay@syneoshealth.com

Guillaume Bonnefois
Syneos Health
5160, boulevard Décarie, Montréal, H3X 2H9
Canada
guillaume.bonnefois@syneoshealth.com

Julie Desrochers
Syneos Health, Québec, Canada
2500, Rue Einstein, Québec, G1P 0A2
Canada
julie.desrochers@syneoshealth.com

Fahima Nekka
Université de Montréal - Faculté de Pharmacie
2940, Chemin de Polytechnique, Montréal, H3T 1J4
Canada
fahima.nekka@umontreal.ca