Paper RE04

Enhancing the Validation and Simulation of Real-World Multi-Site Data with R

Nikolay Trankov, IQVIA, Sofia, Bulgaria
Elena Chaparova, IQVIA, Sofia, Bulgaria
Alexandrina Lambova, IQVIA, Sofia, Bulgaria
Stavros Oikonomou, IQVIA, Sofia, Bulgaria
Monika Petkova, IQVIA, Sofia, Bulgaria
Louise Raiteri, IQVIA, London, United Kingdom

ABSTRACT

Real-world data (RWD) encompasses a broad range of sources, including databases, registries, and electronic health records (EHR). Data from EHR is frequently sourced across multiple hospitals situated in different countries. Due to differences between practices and regions, data often comes in a variety of formats. This data is collected based on a common data specification (CDS). We have developed a standardised format of the CDS, containing definitions of the variables, logic checks and data constraint rules. We have additionally created an R package, which uses the CDS to validate and simulate data. The package reads the CDS and produces outputs of patients and their data fields which fail the logic checks and rules. Furthermore, data can be simulated for the purpose of writing programming code in parallel with the data curation and validation process, which leads to shorter timelines in Real-world evidence studies.

INTRODUCTION

The proliferation of real-world data (RWD) in healthcare research has transformed the landscape of evidence generation and is foundational for real-world evidence (RWE) studies, regulatory submissions, and clinical decision-making. RWD can be sourced from various registries, databases or electronic health records (EHR). Data from EHR can often come from multiple hospitals (or hospital networks) across geographies, which we refer to as multi-site data.

MULTI-SITE DATA AND ITS CHALLENGES

Generally across RWE studies, data obtained directly from hospital sites is considered to have the highest breadth (i.e. variable availability), due to the inherent nature of the collection process. Additionally, it establishes a communication channel with the data entry personnel, which can be leveraged to query any data quality related questions. However, the heterogeneity of data formats, variable definitions, and coding standards across sites poses significant challenges. Setting up and communicating with each hospital site, if done separately, is a slow and iterative process, which can delay data receipt and further, delay study completion. RWD needs to be validated before use, which when done manually is labour-intensive and prone to errors. The growth in size and increase in detail required from RWE studies and high expectations of RWD lead to the requirement of complex, multi-variable validation rules. Analysing each data source separately leads to doubling of work, which increases inconsistencies between analyses, increases the risk of errors and has many inefficiencies.

TRADITIONAL WORKFLOW

Many of the challenges observed when dealing with multi-site data are caused by the historic way of working. Each site receives individual instructions on data and variable requirements. Data from each site is validated separately and often by hand. When queries on the data arise, they are manually collated and shared with each site for resolution. Data from each site is analysed separately, often by separate teams, based on geographical location. One final report combines all analyses and makes conclusions based on all results.
Each of the steps discussed is repeated based on the number of sites providing data, leading to a large amount of work duplication. Duplication is the main cause of data and analysis errors and is an inefficient way of working. Recognising these issues, we worked to identify the main steps of the workflow, which can be combined and automated.

COMMON DATA SPEC (CDS)

In order to avoid the creation of a site-specific document, containing each variable and their associated timepoints, instructions and validation rules, a single study-specific document was decided to be created, called the Common Data Spec. It is an Excel-based document, each sheet representing a separate data table for analysis. Within each table, the following information on the variables to be extracted can be found:

- Category
    - Contains information related to the type of data the variable might contain, e.g. Administrative, Demographic, Clinical, etc. This information would help to categorise the variables within a table.
- Data variable
    - Contains the name of the variable as it would be referred to in regular speech, e.g. Date of birth. This information makes it easier to refer to variables during regular communication.
- CDM variable code
    - Contains the name of the variable as it would be recorded in the data, e.g. BIRTHDAT. Our standard requires all variables names to be written in all caps.
- Description
    - Contains additional information for each variable, to be used both by the sites when entering the data as a data dictionary.
- Type
    - Describes the kind of data the variable might hold. Data can be either categorical (cat), numeric (num) or date (dat).
- Valid values
    - Describes the only possible values that can be entered for each variable. If the variable is of type “categorical”, the valid values would be the specific categories. If the variable is of type “date”, the valid values would be the type of date to be used e.g. YYYYMMDD. If the variable is of type “numeric”, the logic checks must be consulted.
- Logic checks
    - Contains the data constraint rules, to be followed both by the sites at the point of data entry and by the programmers when performing data validation.

The CDS contains a field for specifying each table’s primary key, which is used to guarantee that every record is distinct, cannot be repeated and cannot be null. For tables that are patient-level i.e. one row equates to one patient, the primary key is typically a single column. For event-level tables, the primary key is typically a combination of several columns. For example, for a treatment-level table, where each row equates to one treatment and each patient can have N number of treatments, the primary key would be Patient ID and Treatment ID. The primary keys for each table, which identify a unique record in the table, are always reported in the header row above the table.
Not every column in the CDS is necessary to a programmer. As seen in Table 1, the columns coloured in gray provide additional information necessary only to the sites entering the data. A programmer performing data validation or writing analysis code would only be interested in the Data variable, CDM variable code, Type, Valid values and Logic checks columns.
By enforcing a common specification, the CDS enables consistent data extraction, regardless of site or country, resulting in all data having the same format. Thus, each subsequent step would be performed on harmonised data, eliminating the need to write separate, site-specific code. Therefore, only one set of validation and analysis code can be written, based on the harmonised data, eliminating work duplication.

Table 1: An extract of the CDS, containing all required columns and a selection of a few variables, adapted from a RWE study

Table key:	PTID, SITEID					
Category	**Data Variable**	**CDM Variable Code**	**Description**	**Type**	**Valid Values**	**Logic Checks**
Administrative	Patient ID	PTID	Study-specific patient ID	cat		Required Unique by SITEID
Administrative	Site ID	SITEID	Study-specific site ID	cat	A B C	Required
Demographics	Date of birth	BIRTHDAT	Patient's date of birth	dat	yyyymm	
Clinical characteristics	Height	HEIGHT	Height at diagnosis in cm	num		> 50 < 250
Clinical characteristics	Diagnosis date	DIAGDAT	Date of breast cancer diagnosis	dat	yyyymmdd	Required >= 01/01/2016 <= 31/12/2023 >= 18 years after BIRTHDAT <= 120 years after BIRTHDAT
Biomarker	Patient tested	TESTED	Was the patient tested for PD-L1	cat	Yes No	
Biomarker	Test result	TESTRES	PD-L1 test status	cat	Abnormal Borderline Wild type	Populated when TESTED = "Yes"

R PACKAGE

Our data-source agnostic R package called "IQVIA R Analysis" [1], which is used for the analysis of data and creation of summarised results, was extended with additional functionality. The S4 class "`DataStruct`", is an abstraction for a data set that defines the metadata for each variable within it:

- `NumVariable`: An existing class, used to represent continuous data (e.g. HEIGHT)
- `CatVariable`: An existing class, used to represent categorical data, with or without hierarchical structure (e.g. TESTRES)
- `DateVariable`: A new class, used to represent date data (e.g. DIAGDAT)

The `DataStruct` is, in essence, a list of variable definitions. Inside the `DataStruct` an argument "`variables`" is declared. It takes as an input a list, which contains the declared numerical, categorical or date variables.

Example DataStruct (R Syntax):

```
data_structure <- DataStruct(
  variables = list(
    CatVariable(
      name = "PTID",
      label = "Patient ID"
    ),
    CatVariable(
      name = "SITEID",
      label = "Site ID",
      categories = list(
        list(1, c("A", "B", "C")))
    ),
    DateVariable(
      name = "BIRTHDAT",
      label = "Date of birth"
```

```
    ),
    NumVariable(
      name = "HEIGHT",
      label = "Height"
    ),
    DateVariable(
      name = "DIAGDAT",
      label = "Diagnosis date"
    ),
    CatVariable(
      name = "TESTED",
      label = "Patient tested",
      categories = list(
        list(1, c("Yes", "No")))
    ),
    CatVariable(
      name = "TESTRES",
      label = "Test result",
      categories = list(
        list(1, c("Abnormal", "Borderline", "Wild type")))
  )
)
```

VALIDATION

The `DataStruct` class has been further extended with the ability to define validation rules for each variable. Validation functions can be added as a list to the slot `validateFunc` of a `CatVariable, DateVariable` or `NumVariable` within the `DataStruct`. We have created multiple validation functions to capture the most common validation rules encountered. Should a user encounter a new, previously undefined rule, they can easily create a new validation function, as they have been designed to be flexible. The functions can be grouped into 4 separate classes:

- Check if a column:
  - Exists
  - Contains specific values or objects
  - Contains/does not contain NA values
  - Has distinct row data
- Compare data in a column with a fixed value:
  - All mathematical comparisons are possible
- Compare data in a column with data in another column:
  - All mathematical comparisons are possible
- Compare if data in a column is:
  - Between or outside specified values
  - Between or outside specified columns
  - Part or not part of a set of values
  - Increasing or decreasing
  - Matching a regex
  - In agreement with a predicate expression

The functions support performing validation checks within one or more grouping variables and on the condition that another expression holds true. Moreover, the **`vals_expr`** function allows for assessment of complex validation rules featuring multiple variables and logical operators.
An example `DataStruct` object with validation functions covering the logic checks and data constraints of the CDS extract from Table 1 is shown below:

Example DataStruct with validation functions (R Syntax):

```
data_structure <- DataStruct(
  variables = list(
    CatVariable(
      name = "PTID",
```

```
  label = "Patient ID",
  validateFunc = list(
    categorical_validator$col_exists(),
    categorical_validator$character_vals(),
    categorical_validator$vals_not_na(),
    categorical_validator$unique_vals(
      by_columns = "SITEID")
  )
),
CatVariable(
  name = "SITEID",
  label = "Site ID",
  categories = list(
    list(1, c("A", "B", "C"))),
  validateFunc = list(
    categorical_validator$col_exists(),
    categorical_validator$character_vals(),
    categorical_validator$vals_not_na(),
    categorical_validator$vals_in_set(
      set = c("A", "B", "C"))
  )
),
DateVariable(
  name = "BIRTHDAT",
  label = "Date of birth",
  validateFunc = list(
    date_validator$col_exists(),
    date_validator$date_vals()
  )
),
NumVariable(
  name = "HEIGHT",
  label = "Height",
  validateFunc = list(
    numeric_validator$col_exists(),
    numeric_validator$numeric_vals(),
    numeric_validator$vals_greater_than_value(50),
    numeric_validator$vals_less_than_value(250)
  )
),
DateVariable(
  name = "DIAGDAT",
  label = "Diagnosis date",
  validateFunc = list(
    date_validator$col_exists(),
    date_validator$date_vals(),
    date_validator$vals_not_na(),
    date_validator$vals_greater_than_or_equal_value(
      value = as.Date("2016-01-01")),
    date_validator$vals_less_than_or_equal_value(
      value = as.Date("2023-12-31")),
    date_validator$vals_expr(
      expr = ~ DIAGDAT >= BIRTHDAT + 18*365.25,
      label = ">= 18 years after BIRTHDAT"),
    date_validator$vals_expr(
```

```
          expr = ~ DIAGDAT <= BIRTHDAT + 120*365.25,
          label = "<= 120 years after BIRTHDAT")
      )
    ),
    CatVariable(
      name = "TESTED",
      label = "Patient tested",
      categories = list(
        list(1, c("Yes", "No"))),
      validateFunc = list(
        categorical_validator$col_exists(),
        categorical_validator$character_vals(),
        categorical_validator$vals_in_set(
          set = c("Yes", "No"))
      )
    ),
    CatVariable(
      name = "TESTRES",
      label = "Test result",
      categories = list(
        list(1, c("Abnormal", "Borderline", "Wild type"))),
      validateFunc = list(
        categorical_validator$col_exists(),
        categorical_validator$character_vals(),
        categorical_validator$vals_not_na(
          mode = "all",
          segments = TESTED ~ c("Yes")),
        categorical_validator$vals_in_set(
          set = c("Abnormal", "Borderline", "Wild type"),
          segments = TESTED ~ c("Yes"))
      )
    )
  )
)
```

As the `DataStruct` is simply a list of variable definitions, they must be applied to the data set in order to validate it. This is performed by interrogating the entire data set against the `DataStruct` rules:

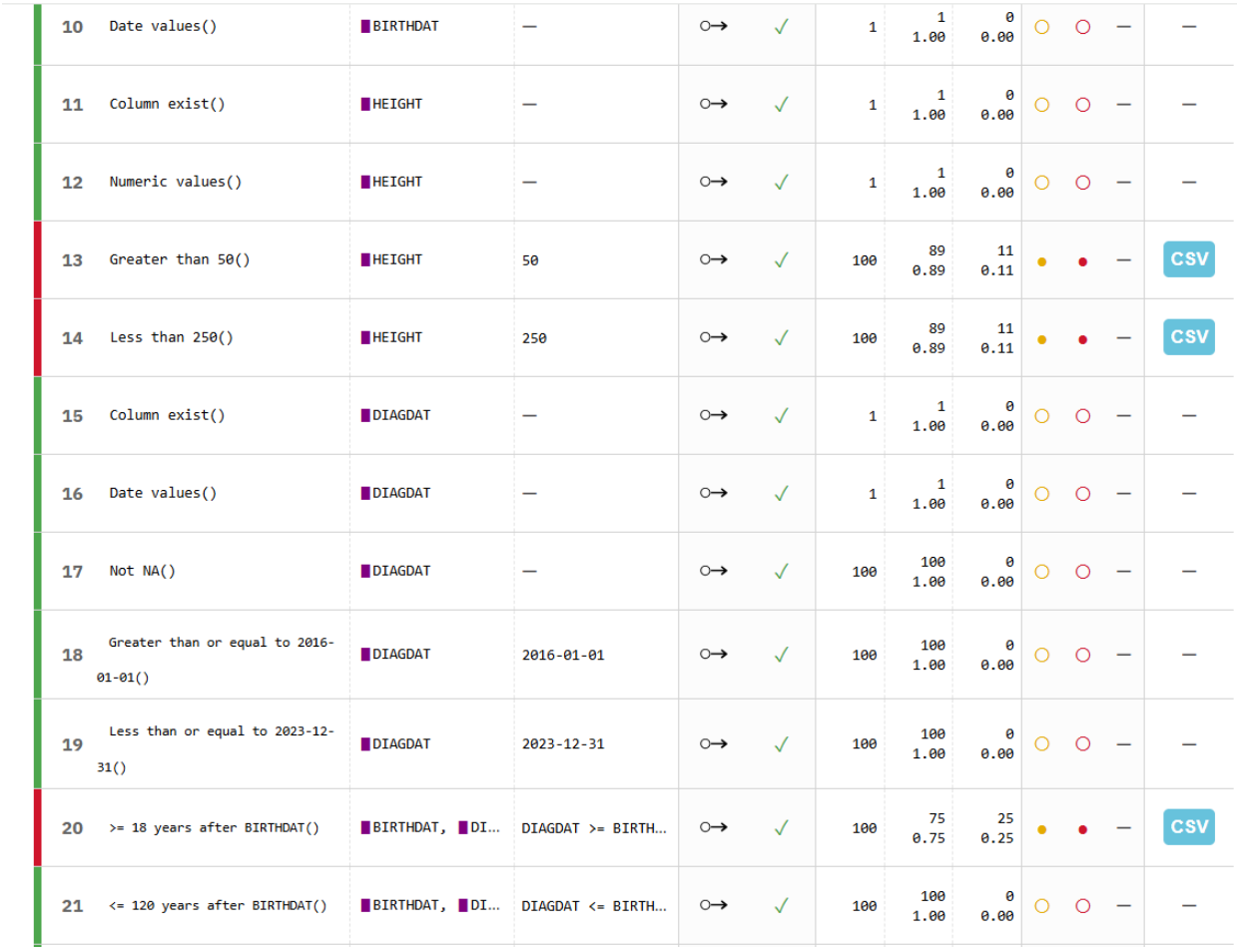
```
validate_agent <- validate(object = data_structure, data = patient_data)
```

A validation report based on the *pointblank* R package [2] can be generated, providing an HTML-based summary of data quality and extracts of row-level data that fail the validation checks. All functionality of the *pointblank* package has been retained and can be applied to the report.

```
pointblank::export_report(validate_agent, filename = "report_patient.html")
```

Each row that fails the validation checks can be easily exported to a .csv file. This file can be used for direct communication with sites, replacing the need for manual collation of queries and reducing time spent on administrative work.

Figure 1: An example validation report



10	Date values()	BIRTHDAT	—	⟶	✓	1	1 1.00	0 0.00	○	○	—
11	Column exist()	HEIGHT	—	⟶	✓	1	1 1.00	0 0.00	○	○	—
12	Numeric values()	HEIGHT	—	⟶	✓	1	1 1.00	0 0.00	○	○	—
13	Greater than 50()	HEIGHT	50	⟶	✓	100	89 0.89	11 0.11	●	●	—
14	Less than 250()	HEIGHT	250	⟶	✓	100	89 0.89	11 0.11	●	●	—
15	Column exist()	DIAGDAT	—	⟶	✓	1	1 1.00	0 0.00	○	○	—
16	Date values()	DIAGDAT	—	⟶	✓	1	1 1.00	0 0.00	○	○	—
17	Not NA()	DIAGDAT	—	⟶	✓	100	100 1.00	0 0.00	○	○	—
18	Greater than or equal to 2016-01-01()	DIAGDAT	2016-01-01	⟶	✓	100	100 1.00	0 0.00	○	○	—
19	Less than or equal to 2023-12-31()	DIAGDAT	2023-12-31	⟶	✓	100	100 1.00	0 0.00	○	○	—
20	>= 18 years after BIRTHDAT()	BIRTHDAT, DI...	DIAGDAT >= BIRTH...	⟶	✓	100	75 0.75	25 0.25	●	●	—
21	<= 120 years after BIRTHDAT()	BIRTHDAT, DI...	DIAGDAT <= BIRTH...	⟶	✓	100	100 1.00	0 0.00	○	○	—

SIMULATION

Once a hospital site has been setup and has received the CDS, data entry begins. This is a process, which can take a substantial amount of time, due to the amount of data to be entered and frequently the limited time available to it. Whilst waiting for data receipt, the study is typically dormant. We have identified this gap in productivity and using our knowledge of working with synthetic data when access to the real data is prohibited [1], have developed a solution to simulate study-specific data based on the CDS. Since the CDS contains all variables to be used in the study and their accompanying information and ensures the consistency of the data format, we can assume that data simulated based on the CDS will have the same properties as the real data once it arrives.
We have developed multiple simulation functions for the different types of data that can be simulated. They are created in a flexible manner and each user can easily create new functions, should they require additional functionality, specificity or a different statistical distribution to be simulated.

- `generate_normal(mean = 0, sd = 1, missing = 0.1)`
  - The objective is to generate samples that conform to a normal distribution.
  - mean - Represents the average value of the samples.
  - sd - Indicates the standard deviation of the normal distribution.
  - missing - Specifies the percentage of missing (NA) values in the sample.
- `generate_uniform(min = 0, max = 1, missing = 0.1)`
  - The objective is to generate samples that conform to a uniform distribution.
  - min - Represents the minimum value of the samples (default: 0).
  - max - Represents the maximum value of the samples (default: 1).
  - missing - Specifies the percentage of missing (NA) values in the sample.
- `generate_range(min = 0, max = 100, missing = 0.1)`

    - The aim is to generate samples with integer values within a certain range, whereby the percentage of missing values can be specified.
    - min - Represents the minimum value of the samples (default: 0).
    - max - Represents the maximum value of the samples (default: 100).
    - missing - Specifies the percentage of missing (NA) values in the sample.
- `generate_nested_sample(categories = list(`
  `list(1, c("Treatment A" = 0.3, "Treatment B" = 0.4)),`
  `list(2, c("Treatment B1" = 0.2, "Treatment B2" = 0.8)),`
  `list(1, c("Treatment C" = 0.2))),`
  `missing = 0.1)`
    - The purpose is to generate nested samples from specified columns. It allows users to define categories and the associated probabilities.
    - categories - Represents list of lists containing the categories to be included in the generated data and their levels. Probabilities also can be specified.
    - missing - Specifies the percentage of missing (NA) values in the sample.
- `generate_dates(from = as.Date("1970-01-01"), to = as.Date("2023-01-01"), by = "year", missing = 0.1)`
    - Generates dates within a specified period and granularity.
    - from - Defines the starting date of the simulation period.
    - to - Specifies the ending date of the simulation period.
    - by - Indicates the granularity of the time intervals used in the simulation.
    - missing - Specifies the percentage of missing (NA) values in the sample.

It is important to note that calling the functions returns a `FunctionCall` object, which contains a simulation function and the arguments to be passed to it. This means that the functions cannot be used on their own the way they are presented here. The simulation functions have been designed to be integrated into the `DataStruct` in order to enable the creation of whole synthetic datasets that accurately replicate the CDS. The declaration of the `DataStruct`, expanded with simulation functions, covering the variables in the CDS extract from Table 1, is shown below:

Example DataStruct with simulation functions (R Syntax):

```
data_structure <- DataStruct(
  variables = list(
    CatVariable(
      name = "PTID",
      label = "Patient ID",
      simulateFunc = generate_nested_sample(
        categories = list(
          list(1, sample(1:10000, 1000, replace = FALSE))),
        missing = 0)
    ),
    CatVariable(
      name = "SITEID",
      label = "Site ID",
      categories = list(
        list(1, c("A", "B", "C"))),
      simulateFunc = generate_nested_sample(
        categories = list(
          list(1, c("A", "B", "C"))),
        missing = 0)
    ),
    DateVariable(
      name = "BIRTHDAT",
      label = "Date of birth",
      simulateFunc = generate_dates(
        from = as.Date("1960-01-01"),
        to = as.Date("2010-12-31"),
        by = "month",
```

```
        missing = 0)
    ),
    NumVariable(
      name = "HEIGHT",
      label = "Height",
      simulateFunc = generate_range(
        min = 50,
        max = 250,
        missing = 0.1)
    ),
    DateVariable(
      name = "DIAGDAT",
      label = "Diagnosis date",
      simulateFunc = generate_dates(
        from = as.Date("2016-01-01"),
        to = as.Date("2023-12-31"),
        by = "month",
        missing = 0)
    ),
    CatVariable(
      name = "TESTED",
      label = "Patient tested",
      categories = list(
        list(1, c("Yes", "No"))),
      simulateFunc = generate_nested_sample(
        categories = list(
          list(1, c("Yes", "No"))),
        missing = 0)
    ),
    CatVariable(
      name = "TESTRES",
      label = "Test result",
      categories = list(
        list(1, c("Abnormal", "Borderline", "Wild type"))),
      simulateFunc = generate_nested_sample(
        categories = list(
          list(1, c("Abnormal", "Borderline", "Wild type"))),
        missing = 0.2)
    )
  )
)
```

As noted previously, the `DataStruct` is simply a list of variable definitions. Thus, in order to create the simulated data, the function "simulate" must be called, which takes as arguments the `DataStruct` and the number of rows to be simulated:

```
simulated_data <- simulate(data_structure, n = 1000)
```

AUTOMATING THE VALIDATION AND SIMULATION

The validation and simulation functions are always based on the CDS rules and definitions. To avoid the need to write code based on predefined rules, the CDS was designed in a format, which is machine-readable. A finite state machine (FSM), represented as a graph in R, parses the CDS logic checks using complex logic and conditional checks. The FSM can be in exactly one of finite number of states at any time. It can change between states based on its encounter of an input [3]. For our purpose, we have used a deterministic FSM, a simplified example of which is shown in Figure 2, where each graph edge corresponds to a valid rule word; as each rule is processed word by word,

the edge paths lead to a single state. Each state returns a specific validation function with arguments drawn from the rule. The list of key words triggering a transition of states is presented in Table 2.

Figure 2: Schematic of the FSM

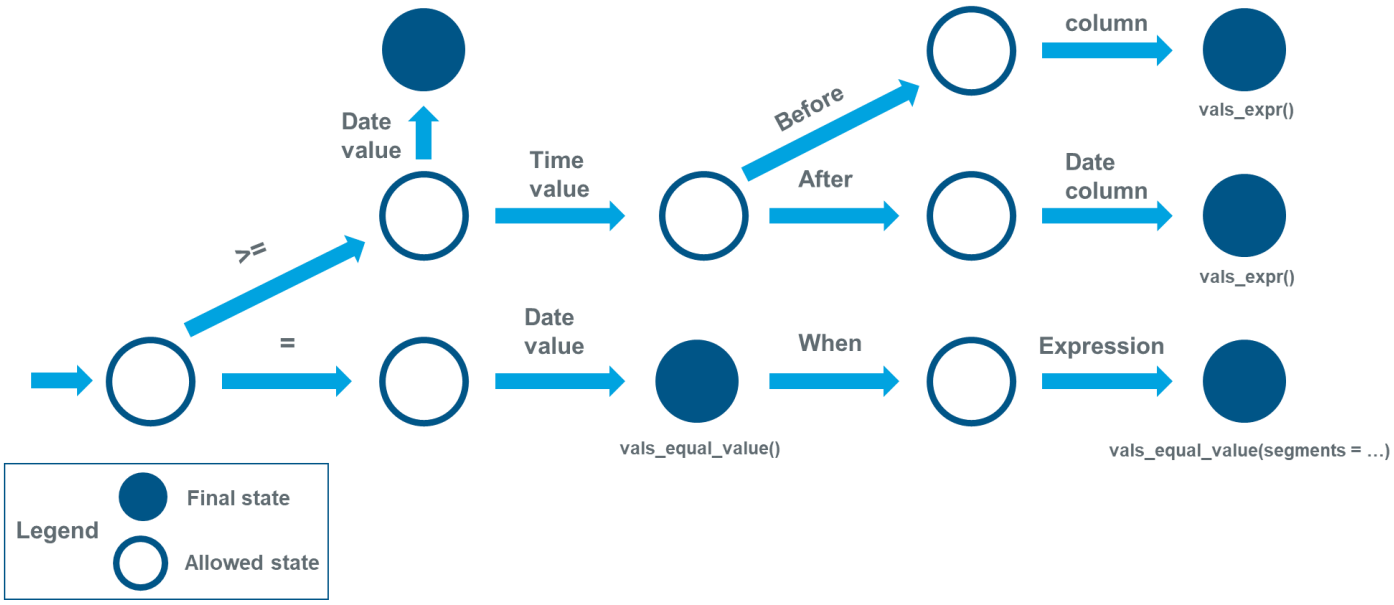


Table 2: List of reserved key words for parsing

Key Word	Explanation
required	Required variables can't have missing values
unique	Values in the variable must be unique
by	The operation will be performed by group based on another column(s)
when	The operation will be performed on the condition that another expression holds true
populated	Variable can't have missing values, used in conjunction with when
or, and, not	Standard logical operators
years, months, days	Defines a unit of time
<, >, <=, >=	Standard operators for date or numeric comparisons
<>, =	Standard operators for all variable types
in ("val1", "val2")	Value is in set

DISCUSSION

Our suggested approach modifies the traditional workflow from site-specific, fragmented and non-optimised validation and analysis code to a singular approach, resulting in harmonised data at the point of data collection and single code written for all data validation and analyses of a study. This greatly reduces the duplication of work and with it the number of errors within both the data and analysis code. The simulation of data to allow for parallel development of code whilst waiting for data receipt removes the bottleneck and allows for the acceleration of study timelines. Integrating the validation and simulation within the already existing IQVIA R Analysis package, allows for easy adoption by users who have previous familiarity with it. Additionally, by automating the CDS reading and generation of simulation and validation functions from it allows for scalability and reduction of errors caused by writing of code by hand. Finally, the creation of automated validation reports eases the communication of various data queries with the sites.

All of the benefits that stem from this automation rely on accurate development and strict adherence to the CDS. If there is an error in the definition of a variable, it would lead to an inaccurate parsing of the simulation and validation functions. If a site does not follow the CDS, additional site-specific data harmonisation must be performed in order to continue the workflow.

Future Work

The work discussed in this paper is a method to streamline and improve an existing and inefficient process. Although CDS is designed to equalise the data across hospital sites, the process of integrating other RWD sources such as registries or databases within a single study has not yet been properly considered. The combination of multiple types of data sources has been proven beneficial in enriching the data [4], even though there are differences within the availability of variables and amount of follow-up.

The use of electronic case report forms (eCRFs) is another way to improve the process of working with multiple sites. eCRFs will forego the need of manual data entry in a tabular form by the sites, since they provide a web interface allowing the completion of questions and capturing the underlying data in a more human-friendly manner. They also eliminate the need for much of the validation, as it will be done at the point of data collection, ensuring greatly improved data quality.

REFERENCES

[1] Lambova, A., Trankov, N., Chaparova, E., Oikonomou, S., Raiteri, L. (2024) Automating Real-World Data Analysis Using R Packages and Shiny: A Three-Stage Workflow, PHUSE EU Connect, https://phuse.s3.eu-central-1.amazonaws.com/Archive/2024/Connect/EU/Strasbourg/PAP_RE06.pdf

[2] Iannone R, Vargas M, Choe J (2025). pointblank: Data Validation and Organization of Metadata for Local and Remote Tables. R package version 0.12.2.9000, https://rstudio.github.io/pointblank/.

Merging Electronic Health Record Data and Genomics for Cardiovascular Research | Circulation: Cardiovascular Genetics

[3] Wang, Jiacun (2019). Formal Methods in Computer Science. CRC Press. p. 34. ISBN 978-1-4987-7532-8.

[4] Hall, J.L. *et al.* (2016) 'Merging Electronic Health Record Data and genomics for Cardiovascular Research', *Circulation: Cardiovascular Genetics*, 9(2), pp. 193–202. doi:10.1161/hcg.0000000000000029.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Author Name: Nikolay Trankov
Company: IQVIA
Address: 103, "Aleksandar Stamboliyski" Blvd., Sofia Tower (Mall of Sofia), floor 7, 1164 Sofia, Bulgaria
Work Phone: N/A
Email: nikolay.trankov@iqvia.com
Website: http://www.iqvia.com/