# Philips IntelliVue Clinical Information Portfolio (ICIP) Database

## Patient data collection

All patients in UHBW CICU have their physiological and therapeutic data continuously monitored and automatically collected by a high-quality data collection system, the Phillips IntelliVue Clinical Information Portfolio (Philips ICIP). Data is stored hourly on an SQL server at UHBW NHS Foundation Trust after manual validation by a unit nurse. The system is divided into a bedside end of the system (the front-end) which is made up of patient monitoring devices to enable straightforward navigation by clinicians, and a SQL database which stores all patient information (the back-end).

The front-end consists of seven data pipelines by which values may be uploaded to the ICIP system. These are vital signs monitoring systems (e.g., heart rate and blood pressure), haematology and biochemistry laboratories, a near-patient blood gas analyser machine, any ventilator the patient is using, any haemofilter the patient is on, and any direct input from a nurse (updating information about prescription and administration of drugs). Nurses must validate any automatically collected data before it is passed to the back-end SQL server for storage. The database files this data in one of several tables using a pair of identifiers according to the data’s content (namely: patient demographics, census, assessments, lab results, medications, site care, and total balance).

## Database structure and organisation

The Philips ICIP system is primarily used as a clinical charting system and not a uniform information storage system. A clinical charting system delivers information on one or a small number of patients to medical professionals in the unit (the front-end) and therefore the structure of the database which stores patient values (the back-end) is entirely dependent on the clinicians’ requirements surrounding data presentation. Updates and edits are made to create a user-friendly front-end rather than a data storage system which means that which patient variables are stored, and how they are stored, varies over time depending on the clinical teams’ needs and wants. This has led to variables being labelled under different names or in different measurement units during different time periods, with each new variation being stored as a unique variable. Duplicate variables may therefore exist. These variables are not well labelled to differentiate between them and their parent, and information about the parent variable is also not given. Organisational structure is not given to the labels; it may therefore be difficult to tell from the label alone how the values stored under a duplicated variable differ, if at all, from the original. No record is kept of changes made to the back-end system by data management staff, and there are no changelogs in the SQL database.

Converting the back-end from a clinical charting system to a research database to make secondary use of the data can be difficult as a research database requires all patient data to be in the same format before analysis of patients can begin. As the edits to the database have been undocumented and no guide or handbook exists to outline where variables are stored, I created a pipeline to extract patient data from the back-end database and process it into a research database. The pipeline allows exploration of the database, examination of the data dictionaries and variable labels, and retrieval of patient values. Basic inspection and cleaning of the values is then performed, at which point values from specific time points throughout a patient’s stay can be extracted to develop a research database.

## Data dictionaries and database organisation

Patient data is filed to one of several tables using a pair of numeric identifiers (“interventionId” and “attributeId”). Both identifiers have their own independent data dictionary which details how the identifier is labelled and described on the front-end for healthcare professionals (Figure 1). Each dictionary has three labels for the identifiers: a *short label*, a *long label*, and a *concept label.* The short label is a truncated form of the long label, and the concept label categorises the Ids together (for example *observable entity* or *qualifier value*). The two data dictionaries are separate entities, and there is no additional information available about how often any two Ids are combined and used in a table. The data dictionaries contain the identifiers for three other ICUs within UHBW as well as the CICU (general, paediatric, and neonatal).

Figure 1. A sample of the interventionId data dictionary. The short label is presented on the front-end when values are being entered to the system. The long label is available if a user hovers over the short label with a mouse and gives the full context for the variable. The concept label helps categorize and organize the variable.

Graphical user interface, text, application

Description automatically generated

The combination of the two identifiers allows for multiple pieces of information to be recorded in relation to a single healthcare procedure. For example, each administration of the drug infusion *Labetalol* has information about the amount, concentration, dose, formulation, volume administered, drug infusion site, and nurse’s comment, each with their own unique attributeId (Figure 2).

Figure 2. A query for “labetalol mg/hr” in the interventionId short label shows how multiple pieces of information can be attached to each administration of the drug. The relevant short labels are shown rather than the interventionId and attributeIds themselves for clarity.

A picture containing text, cabinet

Description automatically generated

The system is designed to be flexible; adding a new variable (i.e., a new drug) is easily done from the front-end. Variables can also be edited by a front-end user to change how they are presented or named (e.g., *Heart Rate* to *HR*). When a variable is edited by a front-end user a new Id is made, and all further information is stored under this Id. The database does not account for duplicates, and so multiple Ids with identical labels can be created.

Extensive un-documented editing to the database has been made to maintain its status as a clinical charting system. Identifiers have often been duplicated in the dictionaries and therefore one variable’s values are split amongst several identifiers in the patient data tables. To continue the earlier example of administration of *Labetalol mg/hr* (interventionId: 14332), there are 8 duplicated attributes for *amount* used in combination with this labetalol interventionId, each with a unique attributeId. This problem extends to the other attributes as well (concentration [8 duplicates], dose [7], formulation [16], volume administered [8], rate administered [3], and drug infusion site [6]).

### Database querying issues

Rather than following precise Ids to retrieve certain variables, wider and more general searches of the labels must be performed to retrieve all the related Ids. The variables with labels similar to the search term must then be manually examined to ensure only the relevant variables are selected. Values entered under each Id are always stored in the same unit of measure, but this is not necessarily uniform between duplicated Ids, meaning values from each duplicated Id combination need to be manually checked for consistency even if the labels are identical.

Searching via labels is also problematic as undocumented front-end editing has led to variations in naming convention according to personal preference, for example the *sodium* bio-chemistry lab test has been changed to both to *salt* and *Na*, and has been cycled through variations of these several times. These labelling problems mean variables can easily be missed during data extraction in a manner that is hard to differentiate from missingness due to clerical or machine error.

In addition to Id duplication and editing, newer Ids do not always follow the style set out when the database was created. Rather than combine an interventionId and attributeId to allow several pieces of information be attached to one event, the information is combined and duplicated across both Ids. Not adhering to the style set out when the database was created has also led to inconsistencies in the use of indicator variables, with the data value being moved between the Id labels and the value entry columns over the course of several Id changes. For example, some drug infusions have been altered to be an indicator variable, reading as *[drug] administered at 4ml/hr* in both the intervention and attributeId labels, with a value of 1 to indicate a patient receiving treatment. To remain in keeping with the rest of the database, this variable should be labelled with the drug as the interventionId label, *administered* as the attributeId label, and then the rate of administration as the value.

The above examples are only some of the ways the database’s management creates difficulties for widespread data extraction for all patients and are not an exhaustive list. The examples serve to illustrate that there is no automated way to retrieve all patient data, and that data extraction for each variable requires attention to detail, good data inspection and analysis via the provided toolkit, and a clinical contact to ensure feasible and realistic output.

## Patient data storage

Patient data was found to be stored in seven tables according to its content: All tables were prefixed with *Pt*, which stood for *patient*. The seven tables were: *demographics*, *census*, *assessments*, *lab results*, *medications*, *site care*, and *total balance*. All patient data tables used intervention and attribute Ids to organise their data except for *total balance*, which only used interventionIds.

The following sixteen columns were considered relevant to data extraction for these tables:

patientId: A patient identifier unique to the ICIP database, analogous to a UHBW T-Number.

encounterId: A patient stay identifier unique to each patient. Used in conjunction with patientId to determine the number of times admitted to the unit.

interventionId: One of two unique identifiers assigned to each variable in the SQL database. interventionIds cover events that occur to the patient: behavioural state, pacer settings, medication, lab tests, etc. Not the recorded piece of information, but the first half of its descriptor.

attributeId: One of two unique identifiers assigned to each variable in the SQL database. attributeIds cover information surrounding an intervention, the *location* of an IV site, the *dosage* of a medication. Not the recorded piece of information, but the second half of its descriptor.

shortLabel: How an interventionId or attributeId is presented on the front-end. Presented as interventionShortLabel/attributeShortLabel.

longLabel: Further information about an interventionId or attributeId, often expanding acronyms. Presented in the same manner as shortLabel (e.g., interventionLongLabel).

conceptLabel: Used to group interventionIds or attributeIds together. Presented in the same manner as shortLabel.

chartTime: The timestamp that the piece of information was entered onto the system in the format YYYY-MM-DD HH:MM:SS

terseForm: A value entry column. Manually entered categorical variables are stored here, typically entered via a drop-down menu from the front-end (e.g., *Mode BIPAP* – a type of ventilation).

verboseForm: A concatenation of the valueNumber and unitOfMeasure columns. How values are displayed on the front-end alongside its unit of measure (e.g., *24 ml/mg/hr*)

valueNumber: A value entry column. The numerical component of verboseForm is stored here.

unitOfMeasure: The measurement component of verboseForm is stored here.

baseValueNumber: A value entry column. valueNumber is transformed according to a new unit of measure, (e.g., mg/ml becomes mg/litre). If no unitOfMeasure is available, valueNumber is repeated here.

baseUOM: The transformed version of unitOfMeasure.

valueString: Free text string.

hourTotal: Used in the Total Balance table in lieu of terseForm, verboseForm, valueNumber, baseValueNumber, and valueString. Stores the hourly change in a value.

cumTotal: Used in the Total Balance table in lieu of terseForm, verboseForm, valueNumber, baseValueNumber, and valueString. Stores the cumulative change in a value over the course of a patient stay.

## Building the main dictionary

**WORKING FROM HERE, UPDATE IN LINE WITH MOST RECENT SCRIPT**

Before data extraction can take place, a “main” dictionary must first be built. The data dictionary contains 6 key pieces of information:

1. Which tables use which intervention and attribute Id combinations to store their data.
2. The number of patients that have values stored under each Id combination (expressed numerically and as a percentage of the population).
3. The mean number of rows of values per patient.
4. The date range each Id combination covers.
5. The percentage of values present in each value entry column (see 1.4 Patient data storage)
6. The meaning ascribed to the used combinations via their short, long, and concept labels.

Items 1 - 5 may be determined by querying each of the patient data tables for 1) the unique numeric Id combinations stored therein 2) the sum of all unique encounterIds under each combination, and 3) the minimum and maximum date, 4) the number of values stored under each Id combination divided by item 2, and 5) the number of non-NA values stored in each value entry column divided by item 4. To express item 2 as a percentage of the population, further validation of patient stays in the unit must be performed (see section X). Item 6 is retrievable via the intervention and attribute data dictionaries.

The resulting data frame contains the following twenty-one pieces of information for each unique combination of Ids used in the database (Table 1):

1 – 2: The intervention and attribute Id number

3 – 4: The number of unique stays in the unit, expressed numerically, and as a percentage of all validated stays.

5: The mean number of rows per patient

6 – 11: The respective short, long, and concept labels for the interventionId and attributeId. I group these by label type as they are read in pairs.

12 – 18: The percentage of each value entry column that contains non-NA values.

19 – 20: The earliest and latest timestamp for the Id combination.

21: The table that the variable is stored under.

Once extracted, the main data dictionary contains around 120,000 Id combinations. Around 80,000 (66.6%) of these Id combinations were used to store information for one patient, although not necessarily the same patient for each combination. A further 20,000 (16.6%) of these Id combinations were used to store information for 2 – 5 patients, and the final 20,000 (16.6%) Id combinations were used to store information for 6 or more patients. Of these 20,000 Id combinations, around 3,400 (~17%) were used for 100 – 1,000 patients, around 1,100 (5.5%) were used for 1,000 – 6,000 patients, and 110 (0.5%) were used for more than 6,000 patients.

Ids used for five or fewer patients in the CICU were removed to ease the future process of searching for Ids related to specific variables.

Table 1. Ten rows from the main dictionary. Id combination labels are always read intervention first then attribute last. When searching for Id combinations via the data processing toolkit, a user’s search terms are queried for in the short and long labels, with the full row returned to the user for inspection. The “percent” column can be higher than 100% due to the presence of unvalidated patient stays. The main dictionary gives no information on the values stored in the tables other than the number of patients with at least one value stored under the combination. For example, as the database is pseudonymised, all values stored under *patient name* variable have been replaced with “\*\*\*”. The second half of the table may be found on the next page.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ids |  | Unique Stays | | Mean rows | Short Label |  | Long Label |  | Concept Label |  |
| Int\* | Att\* | Numeric | Percent |  | Intervention | Attribute | Intervention | Attribute | Intervention | Attribute |
| 4353 | 16240 | 761 | 11.7% | 1.1 | Cholesterol | Value | Cholesterol | Free Form Lab Test.Value | Laboratory data interpretation (procedure) | Measurement (qualifier value) |
| 807 | 17000 | 7139 | 109.9% | 10.2 | Albumin | Albumin | Albumin | Albumin.Albumin.Albumin | Albumin measurement (procedure) | Albumin measurement (procedure) |
| 712 | 7528 | 3033 | 46.7% | 7.6 | INR | INR | INR | INR.INR.INR | International normalized ratio (observable entity) | International normalized ratio (observable entity) |
| 48970 | 16240 | 144 | 2.2% | 1.8 | Venous pH (TC) | Value | Venous pH (TC) | Free Form Lab Test.Value | Laboratory data interpretation (procedure) | Measurement (qualifier value) |
| 22542 | 43995 | 3920 | 60.4% | 7.8 | Neutrophils | Neutrophils | Neutrophils | Neutrophils.Neutrophils | Laboratory test (procedure) | Laboratory test (procedure) |
| 19505 | 16240 | 5 | 0.1% | 1.6 | Urea | Value | Urea | Free Form Lab Test.Value | Laboratory data interpretation (procedure) | Measurement (qualifier value) |
| 48951 | 16240 | 1009 | 15.5% | 35.4 | Na+ | Value | Na+ | Free Form Lab Test.Value | Laboratory data interpretation (procedure) | Measurement (qualifier value) |

\* Int = Intervention, Att = Attribute

Table 1 continued.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Percentage of Value Entry Columns Filled | | | | | | | Timestamps |  | Table |
| terseForm | verboseForm | valueNumber | baseValueNumber | valueString | hourTotal | cumTotal | Minimum | Maximum |  |
| 100 | 100 | 0 | 0 | 100 | NA | NA | 2014-12-31 14:47:00 | 2019-09-13 14:18:00 | PtLabResult |
| 100 | 100 | 100 | 100 | 0 | NA | NA | 2014-12-30 15:40:00 | 2019-09-20 13:24:00 | PtLabResult |
| 100 | 100 | 100 | 100 | 0 | NA | NA | 2015-02-07 05:15:00 | 2017-02-03 13:16:00 | PtLabResult |
| 100 | 100 | 0 | 0 | 100 | NA | NA | 2019-01-11 05:56:00 | 2019-09-18 22:49:00 | PtLabResult |
| 100 | 100 | 100 | 100 | 0 | NA | NA | 2017-01-29 04:19:00 | 2019-09-20 13:24:00 | PtLabResult |
| 100 | 100 | 0 | 0 | 100 | NA | NA | 2016-09-28 06:08:00 | 2016-10-02 05:22:00 | PtLabResult |
| 100 | 100 | 0 | 0 | 100 | NA | NA | 2019-01-07 12:05:00 | 2019-09-20 16:27:00 | PtLabResult |

## Validating patient stays

The formal admittance and discharge timestamps from the CICU were not used to validate patient stays nor determine patient length of stay as they represent when the manual data entry of the event occurred rather than the event itself. Additionally, erroneous admissions to the unit are present due to manual data entry errors. I therefore validate patient stays on the CICU by the presence of a heart rate measurement, with admission and discharge time determined by the timestamp of their first and last recording as the placement of a heart rate monitor is the first and last event to occur to a patient. I remove patients with more than 36 hours difference between the hospital record of admittance to the CICU and their first heartbeat being measured as this was most likely to be erroneous – admission of patients on the Philips system is automatic as a results of patient movements on the wider UHBW patient Electronic Patient Record System (Medway. System C. Maidstone. UK) Some patients were re-admitted to the CICU during the same hospital admission (<1%). Due to this low rate, I considered each stay as unique. Patient LoS was measured in hours.

# Data extraction, inspection, and cleaning toolkit and subsequent data processing pipeline

I present a brief outline of the data processing pipeline here; further details on the specific functions in the toolkit may be found later in this section. Each section is titled with the aim of the section of the pipeline, with bracketed text naming the related functions which may be found in X. The toolkit consists of several functions which are sequentially implemented to retrieve all values pertaining to a particular variable for all patients. Some tools are a mandatory part of data extraction, while others are optional data cleansing tools which are situationally used depending on the data and other outputs of the pipeline. I examined the front-end with a clinician who works on the unit to create a list of variables to search for. This discussion also covered whether each variable should be recorded for all patients or only some subset of patients, which we noted as standard or discretionary care, respectively.

The data processing pipeline begins with querying the main dictionary’s short and long labels for a specific term. The results of the search are then manually inspected to ensure that all relevant Ids have been found and that the labels are consistent and relevant. The sum of patients should be approximately equivalent to the number of patients treated by the unit. If the variable was only recorded for certain patients, such as particular drug infusions only given under certain circumstances, then the retrieved number of patients was sense-checked by a clinician. Subject to any further manipulations such as removal of irrelevant Ids, the Ids retrieved by the search are then queried for in the appropriate table of the SQL database. The values returned from querying the database must then be checked for errors, with symbols and non-numeric values removed from numeric data. For data uploaded from the near-patient blood gas analyser, I also remove non-arterial values to ensure consistency between patients. If multiple Id combinations have been used in the query, then values should also be checked for consistency, with appropriate transformations performed if the unit of measure differs. If two Id combinations share similar or duplicated labels but contain values from different sources (such as from two different machines on the unit), or both have data for all patients, then the data frame should be split and ascribed to a new variable for each Id combination. The SQL database stores values in one of five columns, and this may differ according to the Id combination used. To ease later extraction, I ensure that all values are stored in one column.

## Retrieving relevant Ids from the main dictionary (IdSearch function)

The main dictionary holds all Id combinations used to store patient data across seven tables in the back-end database. When searching for a specific term, the dictionary is subset to a particular table, and then all Id combinations with a similar short or long label to the term are returned. The function has several optional arguments which relate to any intervention or attribute Ids to omit from the search, whether to include the title case and lowercase version of the term when searching, and whether to delete Ids associated with urine analysis.

The outputs of this function are returned via the list *IdSearchOut*, which has five elements (Figure 3):

1. The search term.
2. The intervention Id and its short label
3. The attribute Id and its short label
4. A cross table of the interventionIds (rows) and attributeIds (columns) alongside their short labels, with each cell displaying the number of patients alongside the mean number of values in parentheses.
5. A data frame of the interventionIds and their associated attributeIds

The function prints three items when running (Figure 4). Firstly, the sum of unique stays for all retrieved Ids. This does not account for overlap between the Id combinations' unique stays (no data retrieval has occurred yet), but is useful as a rough measurement to determine whether several variables have been retrieved (e.g., if this number is 200% or above then this is highly likely). Secondly, the cross table of interventionIds and attributeIds, as this is the main element that requires inspection to ensure that the appropriate Ids are retrieved. Thirdly, a statement concerning where to find further information on the retrieved Ids and how to refine this query, with information on the optional arguments.

Figure 3. The list of outputs from running the IdSearch function. The first element is the search term. The second and third elements are a shortened form of the respective data dictionaries for each Id (IntIdsDetail and AttIdsDetail). The fourth output is a cross table detailing the combinations of the Ids in the queried table. The fifth output is the cross table transformed to a data frame to ease investigation when many Ids are retrieved.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 4. The printed statement outputs of the IdSearch function for the search term *APTT* in the *LabResult* table. The sum of all unique stays for all Ids is presented, followed by the cross table of which Id combinations are in use. Finally, information on how to re-run the query with certain Ids removed is presented.

Text

Description automatically generated

After performing a query for a variable in the main dictionary, I check the Id labels for similarity. I determine why interventionId duplications have occurred, whether due to being paired to two different attribute Ids (e.g., if *labetalol* were searched for, then all the attributeId counterparts would be retrieved, while only the dosage may be required). If a duplication like this has occurred, then I determine whether any of these pairings should be removed from any future variable retrieval and re-run the query using the optional arguments to omit those Ids. The number of patients returned per Id combination and in total across the query should also be examined to ensure that all relevant Ids have been collected. The number of patients for each Id combination may be lower than the number of patients processed by the unit for various reasons, primarily due either to the values being split over multiple Id combinations, or to the variable only being collected for certain patients. Discussions with a clinician can give rough estimates of patient population sizes for each discretionary care variable. While the sum of patients retrieved by the query does not indicate any overlap of patients between combinations, it can help determine whether a search term has covered all variations of a variable by comparing it to the total number of known patients in the database. For example, if a variable has been duplicated (same labels, different Id) twice, and the IdSearch function only returns these three terms (the original and the two duplicates), and the sum of all the unique patients is equal to the number of patients in the database, then there is a good chance that the resulting query will cover all patients. Another possibility is that this is a test given to one third of patients, with the test administered in three different ways, but all recorded under the same labels due to poor database management.

## Querying the database for values associated with the retrieved Ids (SQLQuery function)

The values associated with the Ids are then retrieved from the appropriate back-end table. The Ids are supplied to the query directly from the output of the IdSearch function, and only CICU values are retrieved. The patient stay, intervention, and attribute identifiers are returned alongside the five value entry columns where data may be stored (terseForm, verboseForm, valueNumber, baseValueNumber, and valueString), the two unit of measure columns (unitOfMeasure, and baseUOM), and a timestamp. If NA is present in all possible data entry columns, the row is automatically removed. I remove values lower than 0 unless specified otherwise from discussions with a clinician. Once the query is complete, the next function (DataFind) automatically runs.

## Inspecting and cleaning the retrieved values (DataFind, NonNumericFinder, ValueCleaner, and QuantileValueNumber functions).

The Id combination in use determines which of the five data entry columns that a value is written to. When using multiple Id combinations to retrieve patient data, the retrieved data will therefore require inspection and possibly manipulation to ensure that all data is held in one column using the same unit of measure. I begin data inspection with the *DataFind* function, which details the number of NA values in each data entry column. A cross table of how many unique patients have data under each Id combination is then made for any data entry column which has at least 20% values to illustrate the spread of data across the Id combinations and determine whether several variables have been returned by the query (Figure 5). Patient data is moved to the *valueNumber* column for later extraction.

Figure 5. The output of the DataFind function when examining the APTT (activated partial thromboplastin time) query data. Each of the five data entry columns are checked for data values. For any column with more than 20% of its rows filled with values, a cross table of the Id combinations alongside their short labels is presented, provided that the column does not contain a duplicate amount of data as a previous column. Here, a cross table is made for the *terseForm* and *valueString* columns. The cross table for verboseForm is not printed as it has an identical number of values to terseForm. Finally, a final cross table showing unique patients per Id combination for all columns, with interventionId and shortlabel on the rows, and attributeId and shortlabel on the columns.

Text

Description automatically generated

To aid explanation of the data inspection steps, I here describe the steps I took with the data presented in Figure 5. Values in Figure 5 are primarily stored in the terseForm and valueString columns. Values in terseForm were also repeated in verboseForm. I inspected cases where values were not identical across all columns and decided which was the most plausible value (e.g., the row where terseForm was NA was investigated and found to be a blank entry in valueString; the row was subsequently removed from further analysis). I then moved all values to the *valueNumber* column to maintain uniform data frames across variables ease later extraction.

The *NonNumericFinder* function detects any symbols and non-numeric values, which can then be removed via the *ValueCleaner* function (Figure 6). In the case of categorical data, where responses are similar but not identical between Id combinations (*Left ventricular ejection fraction* has both *good* and *good >50%*), I re-assigned values to ensure consistency.

Figure 6. Output of the *NonNumericFinder* function. The APTT values which could not be parsed as numeric alongside their frequency in the variable extraction are returned to the user for inspection and cleansing.

Text

Description automatically generated

The cleaned values may now be visually inspected to ensure value consistency across the Ids and ensure that multiple variables have not been accidentally retrieved. Deciles of the data values for each Id combination are presented alongside Ids themselves, their short labels, the number of unique patients, the unit of measure, and whether the variable is an indicator variable. This output is sorted by the interventionId (Table 2; Figure 7).

One variable’s values may be spread amongst multiple Id combinations due to Id duplicates. The deciles should be visually checked and compared to determine if they are feasibly from the same distribution. If they appear to be from the same distribution, then no further action is needed. If they do not appear to be from the same distribution, or the unit of measure has changed between Ids, then a transformation may be required to ensure consistency for all patients. Splitting the retrieved data frame into two separate variables is also an option if it is likely that the retrieved Ids cover two tests of the same variable. The recorded unit of measure should be consistent for each variable.

Table 2. Output of the QuantileValueColumn function, which is used to determine if multiple similar or duplicated Ids store the same variable by comparing deciles of their values. Additional information is provided about the number of patients with values under each Id combination, the unit of measure recorded (if any), ­­and whether the variable is an indicator or not.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Intervention | | Attribute | |  |  |  | Deciles | | | | | | | | | | |
| Id | Short Label | Id | Short Label | Unique Patients | Unit Of Measure | Indicator Variable | 0% | 10% | 20% | 30% | 40% | 50% | 60% | 70% | 80% | 90% | 100% |
| 800 | APTT | 20461 | APTT | 170 | s | No | 20.8 | 24.5 | 25.9 | 27.1 | 28.4 | 29.5 | 31 | 32.6 | 34.6 | 38.12 | 59.7 |
| 4655 | APTT | 25563 | APTT | 3059 | NA | No | 1 | 24 | 25.5 | 26.6 | 27.8 | 29 | 30.5 | 32.4 | 35.6 | 45.9 | 210 |
| 19515 | APTT | 16240 | Value | 37 | NA | No | 20.5 | 24.1 | 25.4 | 26.5 | 27 | 27.8 | 29.8 | 32.7 | 40.2 | 56.3 | 139.4 |
| 22543 | APTT | 43963 | APTT | 3898 | NA | No | 0.9 | 22.8 | 24.2 | 25.3 | 26.4 | 27.4 | 28.7 | 30.3 | 33 | 40.3 | 200 |

Figure . Second output of the QuantileValueNumber function. The deciles from Table 2 are presented here for ease of inspection. While deciles do not necessarily need to be identical, they should be plausibly similar to ensure that they are referring to the same underlying variable.

Chart, line chart

Description automatically generated

## Removing non-arterial blood gas data (BloodGasArterial function)

The near patient blood gas analyser analyses patients’ blood within the CICU for rapid analysis to aid patient care. Value distributions for the same variable are different for each reading location. After discussion with a clinician, I used *arterial* blood gas readings (rather than *venous* or *other*) as arterial readings provide the most accurate data regarding a patient’s respiratory function. The back-end database stores the information on where a sample type is taken from separately to the values of the reading. I joined the two together, and checked what portion of all readings were arterial, how many patients have arterial data, and how many arterial recordings the patients have on average to ensure that a feasible dataset was produced.

## Lessons about data extraction in Philips ICIP systems.

This toolkit and processing pipeline assumes that the user has little or no prior knowledge of the back-end database. After initial exploration of the database tables to determine where patient data is stored, an immediate priority is building the main dictionary. The main dictionary allows a data analyst to determine what Id combinations are used to store patient variables, and how many patients have values stored under the used combinations. From the main dictionary, even without a clinical contact, it is possible develop an initial dataset by identifying and extracting any variables that are recorded for at least 80% of the patients.

Labels should be heavily scrutinised and investigated when extracting data. Labels can help identify when a variable is measured in multiple ways, especially if the values are spread across multiple tables in the database. For two or more similar or duplicated labels, it is difficult to be entirely certain about whether or not the label was changed to indicate personal preference or two measurement procedures. The only way to be totally certain is from discussion with a clinical contact or through investigation of the front-end. I here highlight three key points where a clinical contact should be consulted to give insight on the data extraction process. These discussions with my supervisor, Dr. Gibbison, were incredibly valuable to me to generate a feasible dataset that covers as many variables as possible.

The first consultation should occur before the back-end database has been accessed and is primarily concerned with how the front-end presents variables to a clinician. Establishing how the front-end of the database is organised and presented will help build a list of variables to search for in the back-end. Investigating the front-end will also help organise variables according to whether the variable is a part of standard or discretionary care (given to all patients or some subset due to their needs). Variables which are measured in multiple ways can be highlighted, and how to differentiate between the two measurements can be explained by the clinician. Further questions about preferred unit of measure per variable will help identify whether data needs processing when extracted from the back-end.

The second clinical consultation should take place after the main dictionary has been built and the first data extraction has been run, but before any specific time point extraction is performed. This discussion should focus on variable processing. Categorical variables may need further refinement, such as conversion from surgery codes to the actual procedure taking place, and the clinical contact can help give meaning to raw data. Transformations which are unintuitive for a data analyst may also be discussed, such as when the unit of measure has changed over time (e.g., mcg/kg/hr for some patients and ml/hr for others).

The third clinical consultation should occur after running data extraction at a specific time point. If multiple sources are available for a variable, whether one should be prioritised should be discussed (e.g., arterial vs venous readings). Additionally, whether one variable should be discarded due to a lack of accuracy compared to other sources should also be considered. Any outstanding issues with missing values may also be discussed, with the clinical contact giving insight as to when it is appropriate to replace missing values with another reading, such as peripheral temperature in lieu of central temperature.