Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

ricu: R's Interface to Intensive Care Data

Nicolas Bennett[*]
ETH Zürich

Drago Plečko[*]
ETH Zürich

Ida-Fong Ukor
Monash Health

Nicolai Meinshausen
ETH Zürich

Peter Bühlmann
ETH Zürich

Abstract

Providing computational infrastructure for handling diverse intensive care unit (ICU) datasets, the R package **ricu** enables writing dataset-agnostic analysis code, thereby facilitating multi-center training and validation of machine learning models. The package is designed with an emphasis on extensibility both to new datasets as well as clinical data concepts, and currently supports the loading of around 100 patient variables corresponding to a total of 395,941 ICU admissions from five data sources collected in Europe and the United States. By allowing for the addition of user-specified medical concepts and data sources, the aim of **ricu** is to foster robust, data-based intensive care research, allowing the user to externally validate their method or conclusion with relative ease, and in turn facilitating reproducible and therefore transparent work in this field.

Keywords: electronic health records, computational physiology, critical care medicine.

1. Introduction

Collection of electronic health records has seen a significant rise in recent years (Evans 2016), opening up opportunities and providing the grounds for a large body of data-driven research oriented towards helping clinicians in decision-making and therefore improving patient care and health outcomes (Jiang, Jiang, Zhi, Dong, Li, Ma, Wang, Dong, Shen, and Wang 2017).

One example of a problem that has received much attention from the machine learning community is early prediction of sepsis in the intensive care unit (ICU; Desautels, Calvert, Hoffman, Jay, Kerem, Shieh, Shimabukuro, Chettipally, Feldman, Barton *et al.* 2016; Nemati, Holder, Razmi, Stanley, Clifford, and Buchman 2018; Futoma, Hariharan, Sendak, Brajer, Clement,

[*]These authors contributed equally.

Bedoya, O'Brien, and Heller 2017; Kam and Kim 2017). Interestingly, there is evidence that a large proportion of the publications are based on the same dataset (Fleuren, Klausch, Zwager, Schoonmade, Guo, Roggeveen, Swart, Girbes, Thoral, Ercole, Hoogendoorn, and Elbers 2020), the Medical Information Mart for Intensive Care III (MIMIC-III; Johnson, Pollard, Shen, Li-wei, Feng, Ghassemi, Moody, Szolovits, Celi, and Mark 2016), which shows a systematic lack of external validation. This issue has recently again been highlighted by a study demonstrating poor performance in external validation of a widely adopted proprietary sepsis prediction model (Wong, Otles, Donnelly, Krumm, McCullough, DeTroyer-Cooley, Pestrue, Phillips, Konye, Penoza, Ghous, and Singh 2021).

Contributing to this problem might well be the lack of computational infrastructure handling multiple datasets. The MIMIC-III dataset consists of 26 different tables containing about 20GB of data. While much work and care has gone into data preprocessing in order to provide a self-contained ready-to-use data resource with MIMIC-III, seemingly simple tasks such as computing a Sepsis-3 label (Singer, Deutschman, Seymour, Shankar-Hari, Annane, Bauer, Bellomo, Bernard, Chiche, Coopersmith, Hotchkiss, Levy, Marshall, Martin, Opal, Rubenfeld, van der Poll, Vincent, and Angus 2016) remain non-trivial efforts[1]. This is only exacerbated when aiming to co-integrate multiple different datasets of this form, spanning hospitals and even countries, in order to capture effects of differing practice and demographics.

The aim of **ricu** is to provide computational infrastructure allowing users to investigate complex research questions in the context of critical care medicine as easily as possible by providing a unified interface to a heterogeneous set of data sources. The package enables users to write dataset-agnostic code which can simplify implementation and shorten the time necessary for prototyping code querying different datasets. In its current form, the package handles five large-scale, publicly available intensive care databases out of the box: MIMIC-III from the Beth Israel Deaconess Medical Center in Boston, Massachusetts (BIDMC; Johnson *et al.* 2016), the eICU Collaborative Research Database (Pollard, Johnson, Raffa, Celi, Mark, and Badawi 2018), containing data collected from 208 hospitals across the United States, the High Time Resolution ICU Dataset (HiRID) from the Department of Intensive Care Medicine of the Bern University Hospital, Switzerland (Faltys, Zimmermann, Lyu, Hüser, Hyland, Rätsch, and Merz 2021), the Amsterdam University Medical Center Database (AmsterdamUMCdb) from the Amsterdam University Medical Center (Thoral, Peppink, Driessen, Sijbrands, Kompanje, Kaplan, Bailey, Keseciogl u, Cecconi, Churpek, Clermont, van der Schaar, Ercole, Girbes, Elbers, on behalf of the Amsterdam University Medical Centers Database (AmsterdamUMCdb) Collaborators, and the SCCM/ESICM Joint Data Science Task Force 2021) and MIMIC-IV, again using data from BIDMC (Johnson, Bulgarelli, Pollard, Horng, Celi, and Mark 2021). Furthermore, **ricu** was designed with extensibility in mind such that adding new public and/or private user-provided datasets is possible. Being implemented in R, a programming language popular among statisticians and data analysts, it is our hope to contribute to accessible and reproducible research by using a familiar environment and requiring only few system dependencies, thereby simplifying setup considerably.

[1]There is considerable heterogeneity in number of patients satisfying the Sepsis-3 criterion (Singer *et al.* 2016) among studies investigating MIMIC-III. Reported Sepsis-3 prevalence ranges from 11.3% (Desautels *et al.* 2016), over 23.9% (Nemati *et al.* 2018) and 25.4% (Wang, Sun, Schroeder, Ameko, Moore, and Barnes 2018), up to 49.1% (Johnson, Aboab, Raffa, Pollard, Deliberato, Celi, and Stone 2018). While some of this variation may be explained by differing patient inclusion criteria, diversity in label implementation must also contribute significantly.

To our knowledge, infrastructure that provides a common interface to multiple such datasets is a novel contribution. While there have been efforts (Adibuzzaman, Musselman, Johnson, Brown, Pitluk, and Grama 2016; Wang, McDermott, Chauhan, Ghassemi, Hughes, and Naumann 2020) attempting to abstract away some specifics of a dataset, these have so far exclusively focused on MIMIC-III, the most popular of public ICU datsets, and have not been designed with dataset interoperability in mind.

Given the somewhat narrow focus of the targeted datasets, it may come as a surprise as to how heterogeneous the resulting datasets are. In MIMIC-III and HiRID, for example, timestamps are reported as absolute times (albeit randomly shifted due to data privacy concerns), whereas eICU and AmsterdamUMCdb use relative times (with origins being admission times). Another example involves different types of patient identifiers and their use among datasets. Common to all is the notion of an ICU admission identifier (ID), but apart from that, the amount of available information varies: While ICU (and hospital) readmissions for a given patient can be identified in some, this is not possible in other datasets. Furthermore, use of identifier systems might not be consistent over tables. In MIMIC-III, for example, some tables refer to ICU stay IDs while others use hospital stay IDs, which slightly complicates data retrieval for a fixed ID system. Additionally, table layouts vary (*long* versus *wide* data arrangement) and data organization in general is far from consistent over datasets.

2. Quick start guide

The following list gives a quick outline of the steps required for setting up and starting to use **ricu**, alongside some section references on where to find further details. A more comprehensive version of this overview is available as a separate vignette.

1. Package installation:

   - Installation from CRAN as `install.packages("ricu")` provides the most recently released version of **ricu**.
   - Alternatively, the latest development version is available from GitHub by running `remotes::install_github("eth-mds/ricu")`.

2. Requesting access to datasets and data source setup:

   - Demo datasets can be set up by installing the data packages `mimic.demo` and/or `eicu.demo` from GitHub using `install.packages()` as shown in Section 3.
   - The complete MIMIC-III, eICU, HiRID and MIMIC-IV datasets can be accessed by registering and setting up a credentialed account at PhysioNet.
   - Access to AmsterdamUMCdb can be requested via the Amsterdam Medical Data Science Website.
   - The obtained credentials can be configured for PhysioNet datasets by setting environment variables `RICU_PHYSIONET_USER` and `RICU_PHYSIONET_PASS`, while the download token for AmsterdamUMCdb can be set as `RICU_AUMC_TOKEN`.
   - Datasets are downloaded and set up either automatically upon the first access attempt or manually by running `setup_data_src()`; the environment variable `RICU_DATA_PATH` can be set to control data location.

- Dataset availability can be queried by calling `src_data_avail()`.

A more detailed description of the supported datasets is given in Section 3, summarized in Table 1, while Section 5 provides implementation details, elaborating on how datasets are represented in code.

3. Loading of data corresponding to clinical concepts using `load_concepts()`:

   - Currently, over 100 data concepts are available for the 4 supported datasets (see `concept_availability()`/`explain_dictionary()` for names, availability etc.).
   - For example, glucose and age data can be loaded by passing `c("age", "glu")` to `load_concepts()`.

   Section 4 goes into more detail on how data concepts are represented within **ricu** and an overview of the preconfigured concepts is available from Section 4.2.

4. Extending the concept dictionary:

   - Data concepts can be specified in code using the constructors `concept()`/`item()` or `new_concept()`/`new_item()`.
   - For session persistence, data concepts can also be specified as JSON formatted objects.
   - JSON-based concept dictionaries can either extend or replace others and they can be pointed to by setting the environment variable `RICU_CONFIG_PATH`.

   The JSON format used to encode data concepts is discussed in more detail in Section 4.3.

5. Adding new datasets:

   - A JSON-based dataset configuration file is required, from which the configuration objects described in Section 5.3 are created.
   - In order for concepts to be available from the new dataset, the dictionary requires extension by adding new data items.

   Further information about adding a new dataset is available from Section 5.4. Some code used when AmsterdamUMCdb was not yet fully integrated with **ricu** is available from GitHub and is used for demonstration purposes to set up AmsterdamUMCdb as an external dataset `aumc_ext`.

Finally, Section 6 shows briefly how **ricu** could be used in practice to address clinical questions by presenting two small examples.

3. Ready-to-use datasets

Several large-scale ICU datasets collected from multiple hospitals in the US and Europe can be set up for access using **ricu** with minimal user effort. Provisions in terms of required

configuration information alongside functions for download and setup are part of **ricu**, opening up easy access to these datasets. Data itself, however, is not part of **ricu** and while the supported datasets are publicly available, access has to be granted by the dataset creators individually. Four datasets, MIMIC-III, MIMIC-IV, eICU and HiRID are hosted on PhysioNet (Goldberger, Amaral, Glass, Hausdorff, Ivanov, Mark, Mietus, Moody, Peng, and Stanley 2000), access to which requires an account, while the fifth, AmsterdamUMCdb is currently distributed via a separate platform, requiring a download link.

For both MIMIC-III and eICU, small subsets of data are available as demo datasets that do not require credentialed access to PhysioNet. As the terms for distribution of these demo datasets are less restrictive, they can be made available as data packages **mimic.demo** and **eicu.demo**. Due to size constraints, however they are not available via CRAN, but can be installed from GitHub as

```
R> install.packages(
+   c("mimic.demo", "eicu.demo"),
+   repos = "https://eth-mds.github.io/physionet-demo"
+ )
```

Provisions for datasets configured to be attached during package loading are made irrespective of whether data is actually available. Upon access of an incomplete dataset, the user is asked for permission to download in interactive sessions and an error is thrown otherwise. Credentials can either be provided as environment variables (`RICU_PHYSIONET_USER` and `RICU_PHYSIONET_PASS` for access to PhysioNet data, as well as `RICU_AUMC_TOKEN` for AmsterdamUMCdb) and if the corresponding variables are unset, user input is again required in interactive sessions. For non-interactive sessions, functionality is exported such that data can be downloaded and set up ahead of first access (see `?setup_src_data`).

Contingent on being granted access by the data owners, download requires a stable Internet connection, as well as 50 to 100 GB of temporary disk storage for unpacking and preparing the data for efficient access. In terms of permanent storage, 5 to 10 GB per dataset are required (see Table 1), while memory requirements are kept reasonably low by iterating over row-chunks for setup operations. Laptop class hardware (8-16 GB of memory) should suffice for setup and many analysis tasks which focus only on subsets of rows (and columns). Initial data source setup (depending on available download speeds and CPU/disk type) may take upwards of an hour per dataset.

The following paragraphs serve to give quick introductions to the included datasets, outlining some strengths and weaknesses of each of the datasets. Especially the PhysioNet datasets MIMIC-III and MIMIC-IV, as well as eICU offer good documentation on the respective websites. Datasets are listed in order of being added to **ricu** and the section is concluded with a table summarizing similarities and differences among the datasets (see Table 1).

3.1. MIMIC-III

The Medical Information Mart for Intensive Care III (MIMIC-III) represents the third iteration of the arguably most influential initiative for collecting and providing large-scale ICU data to the public[2]. The dataset comprises de-identified health related data of roughly 46,000

[2]The initial MIMIC (at the time short for Multi-parameter Intelligent Monitoring for Intensive Care) data

patients admitted to critical care units of BIDMC during the years 2001-2012. Amounting to just over 61,000 individual ICU admission, data is available on demographics, routine vital sign measurements (at approximately 1 hour resolution), laboratory tests, medication, as well as critical care procedures, organized as a 26-table relational structure.

```
R> mimic
```

```
<mimic_env[26]>
        admissions            callout         caregivers        chartevents
     [58,976 x 19]      [34,499 x 24]        [7,567 x 4] [330,712,483 x 15]
         cptevents              d_cpt    d_icd_diagnoses   d_icd_procedures
    [573,146 x 12]          [134 x 9]       [14,567 x 4]        [3,882 x 4]
           d_items         d_labitems     datetimeevents      diagnoses_icd
     [12,487 x 10]          [753 x 6]   [4,485,937 x 14]      [651,047 x 5]
          drgcodes           icustays     inputevents_cv     inputevents_mv
     [125,557 x 8]      [61,532 x 12]  [17,527,935 x 22]   [3,618,991 x 31]
         labevents microbiologyevents         noteevents       outputevents
  [27,854,055 x 9]     [631,726 x 16]   [2,083,180 x 11]   [4,349,218 x 13]
          patients      prescriptions procedureevents_mv     procedures_icd
      [46,520 x 8]   [4,156,450 x 19]     [258,066 x 25]      [240,095 x 5]
          services          transfers
      [73,343 x 6]     [261,897 x 13]
```

One thing of note from a data-organizational perspective is that a change in electronic health care systems occurred in 2008. Owing to this, roughly 38,000 ICU admissions spanning the years 2001 though 2008 are documented using the CareVue system, while for 2008 and onwards, data was extracted from the MetaVision clinical information system. Item identifiers differ between the two systems, requiring queries to consider both ID mappings (heart rate for example being available both as `itemid` number `211` for CareVue and `220045` for MetaVision) as does documentation of infusions and other procedures that are considered as input events (cf., `inputevents_cv` and `inputevents_mv` tables). Especially with respect to such input event data, MetaVision generally provides data of superior quality.

In terms of patient identifiers, MIMIC-III allows for identifying both individual patients (`subject_id`) across hospital admissions (`hadm_id`) and for connecting ICU (re)admissions (`icustay_id`) to hospital admissions. Using the respective one-to-many relationships, **ricu** can retrieve patient data using any of the above IDs, irrespective of how the raw data is organized.

3.2. eICU

Unlike the single-center focus of other datasets, the eICU Collaborative Research Database constitutes an amalgamation of data from critical care units of over 200 hospitals throughout

release dates back 20 years and initially contained data on roughly 100 patients recorded from patient monitors in the medical, surgical, and cardiac intensive care units of Boston's Beth Israel Hospital during the years 1992-1999 (Moody and Mark 1996). Significantly broadened in scope, MIMIC-II was released 10 years after, now including data on almost 27,000 adult hospital admissions collected from ICUs of Beth Israel Deaconess Medical Center from 2001 to 2008 (Lee, Scott, Villarroel, Clifford, Saeed, and Mark 2011).

the continental United States. Large-scale data collected via the Philips eICU program, which provides telehealth infrastructure for intensive care units, is available from the Philips eICU Research Institute (eRI), albeit neither publicly nor freely. Only data corresponding to roughly 200,000 ICU admissions, sampled from a larger population of over 3 million ICU admissions and stratified by hospital, is being made available via PhysioNet. Patients with discharge dates in 2014 or 2015 were considered, with stays in low acuity units being removed.

```
R> eicu
```

```
<eicu_env[31]>
            admissiondrug               admissiondx                   allergy
          [874,920 x 14]             [626,858 x 6]             [251,949 x 13]
             apacheapsvar       apachepatientresult              apachepredvar
          [171,177 x 26]            [297,064 x 23]             [171,177 x 51]
     careplancareprovider               careplaneol           careplangeneral
           [502,765 x 8]               [1,433 x 5]           [3,115,018 x 6]
            careplangoal careplaninfectiousdisease                 customlab
           [504,139 x 7]               [8,056 x 8]               [1,082 x 7]
               diagnosis                  hospital               infusiondrug
         [2,710,672 x 7]                 [208 x 4]           [4,803,719 x 9]
           intakeoutput                       lab                medication
       [12,030,289 x 12]       [39,132,531 x 10]           [7,301,853 x 15]
                microlab                      note            nurseassessment
            [16,996 x 7]          [2,254,179 x 8]           [15,602,498 x 8]
               nursecare            nursecharting                pasthistory
         [8,311,132 x 8]         [151,604,232 x 8]           [1,149,180 x 8]
                 patient             physicalexam           respiratorycare
          [200,859 x 29]           [9,212,316 x 6]            [865,381 x 34]
     respiratorycharting                 treatment             vitalaperiodic
        [20,168,176 x 7]           [3,688,745 x 5]         [25,075,074 x 13]
           vitalperiodic
       [146,671,642 x 19]
```

The data is organized into 31 tables and includes patient demographics, routine vital signs, laboratory measurements, medication administrations, admission diagnoses, as well as treatment information. Owing to the wide range of hospitals participating in this data collection initiative, spanning small, rural, non-teaching health centers with fewer than 100 beds to large teaching hospitals with an excess of 500 beds, data availability varies. Even if data was being recorded at the bedside it might end up missing from the eICU dataset due to technical limitations of the collection process. As for patient identifiers, while it is possible to link ICU admissions corresponding to the same hospital stay, it is not possible to identify patients across hospital stays.

Data resolution again varies considerably over included variables. The `vitalperiodic` table stands out as one of the few examples of a *wide* table organization (laying out variables as columns), as opposed to the *long* presentation (following an entity–attribute–value model) of most other tables containing patient measurement data. The average time step in

`vitalperiodic` is around 5 minutes, but data missingness ranges from around 1% for heart rate and pulse oximetry to around 80-90% for blood pressure measurements, therefore giving approximately hourly resolution for such variables.

3.3. HiRID

Developed for early prediction of circulatory failure (Hyland, Faltys, Hüser, Lyu, Gumbsch, Esteban, Bock, Horn, Moor, Rieck, Zimmermann, Bodenham, Borgwardt, Rätsch, and Merz 2020), the High Time Resolution ICU Dataset (HiRID) contains data on almost 34,000 admissions to the Department of Intensive Care Medicine of the Bern University Hospital, Switzerland, an interdisciplinary 60-bed unit. Given the clear focus on a concrete application during data collection, this dataset is the most limited in terms of data breadth, which is also reflected in a comparatively simple data layout comprising only 5 tables[3].

```
R> hirid
```

```
<hirid_env[5]>
          general      observations           ordinal             pharma
     [33,905 x 5] [776,921,131 x 8]          [72 x 3] [16,270,399 x 14]
        variables
        [712 x 5]
```

Collected during the period of January 2008 through June 2016, roughly 700 distinct variables covering routine vital signs, diagnostic test results and treatment parameters are available with variables monitored at the bedside being recorded with two minute time resolution. In terms of demographic information and patient identifier systems however, the data is limited. It is not possible to identify subsequent ICU admissions corresponding to the same patient and apart from patient age, sex, weight and height, very little information is available to characterize patients. There is no medical history, no admission diagnoses, only in-ICU mortality information, no unstructured patient data and no information on patient discharge. Furthermore, data on body fluid sampling has been omitted, complicating for example the construction of a Sepsis-3 label (Singer *et al.* 2016).

3.4. AmsterdamUMCdb

As a second European dataset, also focusing on increased time-resolution over the US datasets, AmsterdamUMCdb has been made available in late 2019, containing data on over 23,000 intensive care unit and high dependency unit admissions of adult patients during the years 2003 through 2016. The department of Intensive Care at Amsterdam University Medical Center is a mixed medical-surgical ICU with 32 bed intensive care and 12 bed high dependency units with an average of 1000-2000 yearly admissions. Covering middle ground between the US datasets and HiRID in terms of breadth of included data, while providing a maximal

[3]The data is available in three states: as raw data and in two intermediary preprocessing stages explained in Hyland *et al.* (2020). While **ricu** focuses exclusively on raw data, the *merged* stage represents a selection of variables that were deemed most predictive for determining circulatory failure, which are then merged into 18 meta-variables, representing different clinical concepts. Time stamps in *merged* data are left unchanged, yielding irregular time series, whereas for the *imputed* stage, data is down-sampled to a 5 minute grid and missing values are imputed using a scheme discussed in Hyland *et al.* (2020).

time-resolution of 1 minute, AmsterdamUMCdb constitutes a well organized high quality ICU data resource organized succinctly as a 7-table relational structure.

```
R> aumc
```

```
<aumc_env[7]>
         admissions           drugitems       freetextitems
      [23,106 x 17]    [4,907,269 x 31]      [651,248 x 11]
          listitems        numericitems procedureorderitems
  [30,744,065 x 11]  [977,625,612 x 15]     [2,188,626 x 8]
       processitems
      [256,715 x 6]
```

For data anonymization purposes, demographic information such as patient weight, height and age only available as binned variables instead of raw numeric values. Apart from this, there is information on patient origin, mortality, admission diagnoses, as well as numerical measurements including vital parameters, lab results, outputs from drains and catheters, information on administered medication, and other medical procedures. In terms of patient identifiers, it is possible to link ICU admissions corresponding to the same individual, but it is not possible to identify separate hospital admissions.

3.5. MIMIC-IV

The most recently released dataset and next iteration in the MIMIC line of datasets, MIMIC-IV, has recently been released as first stable version (Johnson *et al.* 2021) and support in **ricu** is available as dataset `miiv`. Compared to MIMIC-III, this release shifts focus to newer data, dropping all CareVue-documented patients and with that, patients who were admitted before 2008, while adding patients admitted up to and including 2019. The resulting dataset contains data on over 256,000 patients of which, 53,000 were admitted to ICUs, resulting in 76,000 unique ICU and almost 70,000 related hospital admissions.

```
R> miiv
```

```
<miiv_env[27]>
        admissions        chartevents            d_hcpcs    d_icd_diagnoses
    [523,740 x 15] [329,499,788 x 10]       [89,200 x 4]      [109,775 x 3]
  d_icd_procedures            d_items         d_labitems     datetimeevents
      [85,257 x 3]        [3,861 x 9]        [1,630 x 5]    [7,495,712 x 9]
     diagnoses_icd           drgcodes               emar        emar_detail
   [5,280,351 x 5]      [769,622 x 7]  [27,464,367 x 11]  [55,947,921 x 33]
       hcpcsevents           icustays       inputevents          labevents
     [160,727 x 6]       [76,540 x 8]   [9,460,658 x 26] [122,103,667 x 15]
microbiologyevents       outputevents           patients           pharmacy
  [3,397,914 x 24]    [4,457,381 x 8]      [382,278 x 6]  [14,736,386 x 27]
                poe         poe_detail      prescriptions    procedureevents
 [42,483,962 x 11]    [3,256,358 x 5]  [17,008,053 x 17]     [731,247 x 26]
    procedures_icd           services         transfers
     [779,625 x 6]      [562,892 x 5]    [2,189,535 x 7]
```

Table 1: Comparison of datasets supported by **ricu**, highlighting some of the major similarities and distinguishing features among the five data sources described in the preceding sections. Values followed by parenthesized ranges represent medians and are accompanied by interquartile ranges.

	MIMIC (demo)	eICU (demo)
Number of tables	25	31
Disk storage [GB]	0.01	0.06
Largest table [rows]	758,355	1,634,960
Available concepts*	89	86
Data collection		
Time span	NA	NA
Country of origin	NA	NA
Admission counts		
ICU	136	2,520
Hospital	129	2,174
Unique patients	100	-
Stay lengths [day]		
ICU stays	2.11 (1.23–4.33)	1.47 (0.79–2.78)
Hospital stays	6.63 (3.31–10.65)	4.14 (2.12–7.90)
Vital signs [1/hour]		
Heart rate	1.00 (1.00–1.00)	12.00 (12.00–12.00)
Mean arterial pressure	1.00 (1.00–1.00)	12.00 (4.00–12.00)
Lab tests [1/day]		
Bilirubin	1.02 (0.98–2.01)	1.00 (0.89–1.16)
Lactate	5.00 (1.75–10.75)	3.69 (1.38–6.40)

* These values represent the number of atomic concepts per data source. Additionally, 29 recursive concepts are available, which build on data source specific atomic concepts in a source agnostic manner (see Section 4.3 for details).

Note: The following code blocks are run using demo (`mimic_demo` and `eicu_demo`) instead of full (`mimic_demo` and `eicu_demo`) datasets and therefore might be less useful. For data set up, please consult the manual at `?attach_src`. The full version of this vignette is available from CRAN.

In addition to including newer ICU data, this MIMIC release puts both more emphasis on data collected outside the ICU, newly making emergency department (ED) data available. In a similar vein, the set of considered data types is also expanded by including chest X-ray (CXR) imagery directly with MIMIC data, using the same patient identifiers, while expanding the amount of unstructured text data (still to be made publicly available). Despite these promising developments, the focus of **ricu** remains on data that lies in the intersection of the supported datasets and therefore both ED and CXR data cannot be accessed by the current `miiv` implementation. Finally, documentation of medication administration has been much improved by not only reporting prescriptions, but, using an electronic Medicine Administration Record (eMAR) system, including time-stamped data on administration of individual formulary units.

4. Data concepts

One of the key components of **ricu** is a scheme for specifying how to retrieve data corresponding to predefined clinical concepts from a given data source. This abstraction provides a mechanism for hiding away the data source specific implementation of a concept, in turn enabling dataset agnostic code for analysis. Heart rate, for example can be loaded from datasets `mimic_demo` and `eicu_demo` using the `hr` concept as

```
R> src  <- "mimic_demo"
R> demo <- c(src, "eicu_demo")
R>
R> load_concepts("hr", demo, verbose = FALSE)

# A ‘ts_tbl‘: 152,197 x 4
# Id vars:    ‘source‘, ‘icustay_id‘
# Units:      ‘hr‘ [bpm]
# Index var:  ‘charttime‘ (1 hours)
        source     icustay_id charttime    hr
        <chr>           <int> <drtn>    <dbl>
      1 eicu_demo      141764   0 hours   119
      2 eicu_demo      141764   1 hours   105
      3 eicu_demo      141764   2 hours    95
      4 eicu_demo      141764   3 hours   106
      5 eicu_demo      141764   4 hours   102
    ...
152,193 mimic_demo     298685 314 hours    60
152,194 mimic_demo     298685 315 hours    56
152,195 mimic_demo     298685 316 hours    50
152,196 mimic_demo     298685 317 hours    48
```

```
152,197 mimic_demo     298685 318 hours     0
# ... with 152,187 more rows
```

This requires infrastructure for specifying how to retrieve data subsets (Section 4.3) that is both extensible (to new concepts and new datasets) and flexible enough to handle concept-specific preprocessing. Furthermore, allowing for code re-use for common data transformation tasks is important for simplifying both code development and maintenance. Building on this framework, **ricu** has included a dictionary with over 100 concepts implemented for all five supported datasets (where possible; see also Section 4.2 for further details).

4.1. Data classes

In order to represent tabular ICU data, **ricu** provides several classes, all inheriting from `data.table`. The most basic of which, `id_tbl`, marks one (or several) columns as `id_vars` which serve to define a grouping (i.e., identify patients or unit stays). Inheriting from `id_tbl`, `ts_tbl` is capable of representing grouped time series data. In addition to `id_var` column(s), a single column is marked as `index_var` and is required to hold a base R `difftime` vector. Furthermore, `ts_tbl` contains a scalar-valued `difftime` object as `interval` attribute, specifying the time series step size. More recently, a further class, `win_tbl`, inheriting from `ts_tbl` has been added. Objects of this class can be used for time-stamped measurements associated with a validity period. A set of drug infusions, consisting of both rates and intervals can as such be conveniently represented by a `win_tbl` object.

Metadata for classes inheriting from `id_tbl` is transiently added to `data.table` objects and for S3 generic functions which allow for object modifications, down-casting is implicit:

```
R> (dat <- ts_tbl(a = 1:5, b = hours(1:5), c = rnorm(5)))

# A ‘ts_tbl‘: 5 x 3
# Id var:     ‘a‘
# Index var:  ‘b‘ (1 hours)
      a b             c
  <int> <drtn>    <dbl>
1     1 1 hours  1.77
2     2 2 hours  0.799
3     3 3 hours -1.32
4     4 4 hours -0.0269
5     5 5 hours  1.04

R> dat[["b"]] <- dat[["b"]] + mins(30)
R> dat

# An ‘id_tbl‘: 5 x 3
# Id var:      ‘a‘
      a b                c
  <int> <drtn>       <dbl>
1     1  5400 secs  1.77
```

```
2     2  9000 secs  0.799
3     3 12600 secs -1.32
4     4 16200 secs -0.0269
5     5 19800 secs  1.04
```

Due to time series step size of `dat` being specified as 1 hour, an internal inconsistency is encountered when shifting time stamps by 30 minutes, as time steps are no longer multiples of the time series interval, in turn causing down-casting to `id_tbl`. Furthermore, if column `a` were to be removed, direct down-casting to `data.table` would be required in order to resolve resulting inconsistencies[4].

Coercion to base classes `data.frame` and `data.table`, by stripping away the extra attributes, is easily possible using functions `as.data.frame()` and `as.data.table()`. Coercion is also available as `data.table`-style by-reference operation by passing `by_ref = TRUE` to any of the above coercion functions. User caution is advised, as this does break with base R by-value (or copy-on-modify) semantics and may lead to unexpected behavior.

In its current form, `win_tbl` objects can both be used to represent for example drug rates or drug amounts, administered over a specified time-period. When calling the utility function `expand()` however, which creates a `ts_tbl` from a `win_tbl` by assigning values to the corresponding time steps, values are assumed to be *valid* for the given interval.

```
R> (dat <- win_tbl(a = 1:5, b = hours(1:5), c = mins(rep(90, 5)),
+                  d = runif(5)))

# A ‘win_tbl‘:  5 x 4
# Id var:       ‘a‘
# Index var:    ‘b‘ (1 hours)
# Duration var: ‘c‘
      a b       c           d
  <int> <drtn>  <drtn>  <dbl>
1     1 1 hours 90 mins 0.188
2     2 2 hours 90 mins 0.629
3     3 3 hours 90 mins 0.323
4     4 4 hours 90 mins 0.360
5     5 5 hours 90 mins 0.966

R> expand(dat)

# A ‘ts_tbl‘: 10 x 3
# Id var:     ‘a‘
# Index var:  ‘b‘ (1 hours)
       a b           d
   <int> <drtn>  <dbl>
 1     1 1 hours 0.188
```

[4]Updating an object inheriting from `id_tbl` using `data.table::set()` bypasses consistency checks as this is not an S3 generic function and therefore its behavior cannot be tailored to requirements of `id_tbl` objects. It therefore is up to the user to avoid creating invalid `id_tbl` objects in such a way.

```
 2     1 2 hours 0.188
 3     2 2 hours 0.629
 4     2 3 hours 0.629
 5     3 3 hours 0.323
 6     3 4 hours 0.323
 7     4 4 hours 0.360
 8     4 5 hours 0.360
 9     5 5 hours 0.966
10     5 6 hours 0.966
```

In a case where `d` represented drug amounts instead of drug rates, the current implementation of `expand()` would produce incorrect results. One would expect the overall amount in such a scenario to be evenly divided by – and the resulting fractions assigned to – the corresponding time steps. Allowing for this distinction is being considered, but, as of yet, has not been implemented.

Utilizing the attached metadata of objects inheriting from `id_tbl`, several utility functions can be called with concise semantics (as seen in the above example, where `expand()` is able to determine the required column names from the `win_tbl` object by default). Utilities include functions for sorting, checking for duplicates, aggregating data per combination of `id_vars` (and time step/time duration), checking time series data for gaps, verifying time series regularity and converting between irregular and regular time series, as well as functions for several types of moving window operations. Adding to those class-specific implementations, `id_tbl` objects inherit from `data.table` (and therefore from `data.frame`), ensuring compatibility with a wide range of functionality targeted at these base-classes.

4.2. Ready-to-use concepts

The current selection of clinical concepts that is included with **ricu** covers many physiological variables that are available throughout the included datasets. Treatment-related information on the other hand, being more heterogeneous in nature and therefore harder to harmonize across datasets, has been added on an as-needed basis and therefore is more limited in breadth. A quick note on loading from multiple sources simultaneously: In the introductory example, heart rate was loaded from multiple data sources, resulting in a column `source` being added. This allows for identifying patient IDs corresponding to the respective data sources and the extra column is added to the set of `id_vars`. In the following calls to `load_concepts()`, only data from a single source is requested and therefore no corresponding `source` column is added.

Available concepts can be enumerated using `load_dictionary()` and the utility function `explain_dictionary()` can be used to display some concept metadata.

```
R> dict <- load_dictionary(demo)
R> head(dict)
```

```
<concept[6]>
                                  abx                                  adh_rate
          antibiotics <lgl_cncpt[4]>       vasopressin rate <num_cncpt[3]>
```

```
                                    adm                                     age
patient admission type <fct_cncpt[2]>           patient age <num_cncpt[2]>
                                    alb                                     alp
              albumin <num_cncpt[2]>   alkaline phosphatase <num_cncpt[2]>
```

```
R> explain_dictionary(head(dict))
```

```
      name     category            description
1      abx  medications            antibiotics
2 adh_rate  medications       vasopressin rate
3      adm demographics patient admission type
4      age demographics            patient age
5      alb    chemistry                albumin
6      alp    chemistry   alkaline phosphatase
```

The following subsections serve to introduce some of the included concepts as well as highlight limitations that come with current implementations. Grouping the available concepts by category yields the following counts

```
R> table(vapply(dict, `[[`, character(1L), "category"))
```

```
   blood gas    chemistry demographics   hematology  medications
          10           21            6           20           17
microbiology neurological      outcome       output  respiratory
           1            8           19            2           10
      vitals
           6
```

Physiological data

The largest and most well established group of concepts (covering more than half of all currently included concepts) includes physiological patient measurements such as routine vital signs, respiratory variables, fluid discharge amounts, as well as many kinds of laboratory tests including blood gas measurements, chemical analysis of body fluids and hematology assays.

```
R> load_concepts(c("alb", "glu"), src, interval = mins(15L),
+                verbose = FALSE)
```

```
# A `ts_tbl`: 1,965 x 4
# Id var:     `icustay_id`
# Units:      `alb` [g/dL], `glu` [mg/dL]
# Index var:  `charttime` (15 mins)
      icustay_id charttime    alb   glu
           <int> <drtn>     <dbl> <dbl>
```

```
    1     201006 -3495 mins  NA     116
    2     201006 -2745 mins  NA      83
    3     201006 -1275 mins  NA      91
    4     201006    15 mins   2.4   175
    5     201006   675 mins  NA     129
  ...
1,961     298685 15600 mins  NA     159
1,962     298685 16365 mins   2.2   153
1,963     298685 17400 mins  NA     182
1,964     298685 17595 mins  NA     122
1,965     298685 17955 mins   2.5   121
# ... with 1,955 more rows
```

Most concepts of this kind are represented by `num_cncpt` objects (see Section 4.3) with an associated unit of measurement and a range of permissible values. Data is mainly returned as `ts_tbl` objects, representing time-dependent observations. Apart from conversion to a common unit (using functionality offered by the **units** package (Pebesma, Mailund, and Hiebert 2016) or possibly using the `convert_unit()` callback function), little has to be done in terms of preprocessing: values are simply reported at time-points rounded to the requested interval.

Patient demographics

Moving on from dynamic, time-varying patient data, this group of concepts focuses on static patient information. While the assumption of remaining constant throughout a stay is likely to hold for variables including patient sex or height this is only approximately true for others such as weight. Nevertheless, such effects are ignored and concepts of this group will be mainly returned as `id_tbl` objects with no corresponding time-stamps included.

Whenever requesting concepts which are returned with associated time-stamps (e.g., glucose) alongside time-constant data (e.g., age), merging will duplicate static data over all time-points.

```
R> load_concepts(c("age", "glu"), src, verbose = FALSE)
```

```
# A ‘ts_tbl‘: 1,914 x 4
# Id var:     ‘icustay_id‘
# Units:      ‘age‘ [years], ‘glu‘ [mg/dL]
# Index var:  ‘charttime‘ (1 hours)
      icustay_id charttime   age   glu
           <int> <drtn>    <dbl> <dbl>
    1     201006 -58 hours  68.9   116
    2     201006 -45 hours  68.9    83
    3     201006 -21 hours  68.9    91
    4     201006   0 hours  68.9   175
    5     201006  11 hours  68.9   129
  ...
1,910     298685 260 hours  80.1   159
1,911     298685 272 hours  80.1   153
1,912     298685 290 hours  80.1   182
```

```
1,913     298685 293 hours  80.1   122
1,914     298685 299 hours  80.1   121
# ... with 1,904 more rows
```

Despite a best-effort approach, data availability can be a limiting factor. While for physiological variables, there is good agreement even across countries, data-privacy considerations, as well as lack of a common standard for data encoding, may cause issues that are hard to resolve. In some cases, this can be somewhat mitigated while in others, this is a limitation to be kept in mind. In AmsterdamUMCdb, for example, patient age, height and weight are not available as continuous variables, but as factor variables with patients binned into groups. Such variables are then approximated by returning the respective mid-points of groups for `aumc` data[5]. Other concepts, such as `adm` (categorizing admission types) or a potential `icd` concept (diagnoses as ICD-9 codes) can only return data if available from the data source in question. Unfortunately, neither `aumc` nor `hirid` contain ICD-9 encoded diagnoses, and in the case of `hirid`, no diagnosis information is available at all.

Treatment-related information

The largest group of concepts dealing with treatment-related information is described by the `medications` category. In addition to drug administrations, only basic ventilation information is currently provided as ready-to-use concept. Just like availability of common ICU procedures, patient medication is also underdeveloped, covering mainly vasopressor administrations, as well as corticosteroids, antibiotics and dextrose infusions. The current concepts retrieving treatment-related information are mostly focused on providing data required for constructing clinical scores described in Section 4.2.4. While this group of concepts lends itself to use of `win_tbl` objects, a call to `load_concepts()`, requesting multiple concepts which do not all return data as `win_tbl` (while leaving the `merge` argument at default value `TRUE`), all `win_tbl` objects are converted to `ts_tbl` in order to be merged with the non-`win_tbl` objects.

Ventilation is represented by several concepts: a ventilation indicator variable (`vent_ind`), as well as ventilation durations (`vent_dur`) are constructed from start and end events (concepts `vent_start` and `vent_end`). This includes any kind of mechanical ventilation (invasive via an endotracheal or tracheostomy tube), as well as non-invasive ventilation via face or nasal masks. In line with other concepts belonging to this group, the current state is far from being comprehensive and expansion to further ventilation parameters is desirable.

The singular concept addressing antibiotics (`abx`) returns an indicator signaling whenever an antibiotic was administered. This includes any route of administration (intravenous, oral, topical, etc.) and does neither report dosage, nor active ingredient. Finally, vasopressor administration is reported by several concepts representing different vasoactive drugs (including dopamine, dobutamine, epinephrine, norepinephrine and vasopressin), as well as different administration aspects such as rate, duration and rate administered for at least 60 minutes, which is used in Sepsis-Related Organ Failure Assessment (SOFA) scoring (Vincent, Moreno, Takala, Willatts, De Mendonça, Bruining, Reinhart, Suter, and Thijs 1996).

[5]Prioritizing consistency over accuracy, one could apply the same binning to datasets which report numeric values, but the concepts included with **ricu** attempt to strike a balance between consistency and amount of applied preprocessing. With the extensible architecture of data concepts, however, such categorical variants of patient demographic concepts could easily be added.

```
R> load_concepts(c("abx", "vent_ind", "norepi_rate", "norepi_dur"), src,
+                verbose = FALSE)

# A ‘ts_tbl‘: 12,434 x 6
# Id var:     ‘icustay_id‘
# Units:      ‘norepi_rate‘ [mcg/kg/min]
# Index var:  ‘startdate‘ (1 hours)
       icustay_id startdate abx   vent_ind norepi_rate norepi_dur
            <int> <drtn>    <lgl> <lgl>          <dbl> <drtn>
     1     201006   7 hours TRUE  NA           NA      NA hours
     2     201006   8 hours NA    TRUE         NA      60 hours
     3     201006   9 hours NA    TRUE          0.0460 NA hours
     4     201006  10 hours NA    TRUE          0.0690 NA hours
     5     201006  11 hours NA    TRUE          0.0690 NA hours
   ...
12,430     298685 612 hours NA    TRUE         NA      NA hours
12,431     298685 613 hours NA    TRUE         NA      NA hours
12,432     298685 614 hours NA    TRUE         NA      NA hours
12,433     298685 615 hours NA    TRUE         NA      NA hours
12,434     298685 616 hours NA    TRUE         NA      NA hours
# ... with 12,424 more rows
```

As cautioned in Section 4.2.2, variability in data reporting across datasets can lead to issues: the `prescriptions` table included with MIMIC-III, for example, reports time-stamps as dates only, yielding a discrepancy of up to 24 hours when merged with data where time-accuracy is on the order of minutes. Another problem exists with concepts that attempt to report administration windows, as some datasets do not describe infusions with clear cut start/endpoints but rather report infusion parameters at (somewhat) regular time intervals. This can cause artifacts when the requested time step-size deviates from the dataset inherent time grid and introduces uncertainty when attempting to determine start/endpoints for creating a `win_tbl` object.

```
R> load_concepts("dex", "mimic_demo", verbose = FALSE)

# A ‘win_tbl‘:  10 x 4
# Id var:       ‘icustay_id‘
# Units:        ‘dex‘ [ml/hr]
# Index var:    ‘starttime‘ (1 hours)
# Duration var: ‘dur_var‘
   icustay_id starttime dur_var       dex
        <int> <drtn>    <drtn>      <dbl>
 1     216185 129 hours   1 mins 15000
 2     249805 335 hours 660 mins     9.09
 3     253931 159 hours   1 mins   500.
 4     277238   2 hours 140 mins   100.
 5     277238   3 hours   1 mins 15000
 6     277238   5 hours   1 mins 15000
```

```
 7     285750   4 hours   1 mins 15000
 8     286072 101 hours   1 mins  7500
 9     286072 118 hours   1 mins  7500
10     286072 126 hours   1 mins  7500
```

Furthermore for a concept like dextrose administration as implemented in `dex`, where infusions are returned alongside bolus administrations, this can yield large rate values, as the returned unit is ml/hr and in this particular case, values are harmonized such that they correspond to 10% dextrose solutions. A bolus administration of 50 ml dextrose 50% will therefore be reported as 15000 ml/hr administered within 1 minute.

Outcomes

A group of more loosely associated concepts can be used to describe patient state. This includes common clinical endpoints, such as death or length of ICU stay, as well as scoring systems such as SOFA, the systemic inflammatory response syndrome (SIRS; Bone, Sibbald, and Sprung 1992) criterion, the National Early Warning Score (NEWS; Jones 2012) and the Modified Early Warning Score (MEWS; Subbe, Kruger, Rutherford, and Gemmel 2001).

While the more straightforward outcomes can be retrieved directly from data, clinical scores often incorporate multiple variables, based upon which a numeric score is constructed. This can typically be achieved by using concepts of type `rec_cncpt` (see Section 4.3), specifying the needed components and supplying a callback function that applies rules for score construction.

```
R> load_concepts(c("sirs", "death"), src, verbose = FALSE,
+                keep_components = TRUE)

# A ‘ts_tbl‘: 14,295 x 8
# Id var:     ‘icustay_id‘
# Index var:  ‘charttime‘ (1 hours)
       icustay_id charttime  sirs death temp_comp hr_comp resp_comp wbc_comp
            <int> <drtn>    <dbl> <lgl>     <int>   <int>     <int>    <int>
     1     201006 -58 hours     1 NA           NA      NA        NA        1
     2     201006 -45 hours     1 NA           NA      NA        NA        1
     3     201006 -21 hours     1 NA           NA      NA        NA        1
     4     201006 -10 hours     2 NA           NA      NA         1        1
     5     201006   0 hours     3 NA            0       1         1        1
   ...
14,291     298685 314 hours     2 NA            0       0         1        1
14,292     298685 315 hours     2 NA            0       0         1        1
14,293     298685 316 hours     2 NA            0       0         1        1
14,294     298685 317 hours     1 NA            0       0         0        1
14,295     298685 318 hours     1 TRUE          0       0        NA        1
# ... with 14,285 more rows
```

Callback functions can become rather involved (especially for more complex concepts such as SOFA) and may offer arbitrary arguments to tune their behavior. As callback functions to `rec_cncpt` objects are typically called internally from `load_concepts()`, arguments not used

by `load_concepts()`, such as `keep_components` in the above example (causing not only the score column, but also individual score components to be retained) are forwarded. Therefore, some care has to be taken as when requesting multiple concepts within the same call to `load_concepts()`, while passing arguments intended for concept-level callback functions, as all involved callback functions will be called with the same forwarded arguments. When for example requesting multiple scores (such as SOFA or SIRS), it is currently not possible to enable `keep_components` for only a subset thereof. This setup consequently also requires that all involved callback functions are allowed to be called with the given set of extra arguments.

4.3. Concept specification

Just like data source configuration (as discussed in Section 5.3), concept specification relies on JSON-formatted text files, parsed by **jsonlite** (Ooms 2014). A default dictionary of concepts is included with **ricu**, containing a selection of commonly used clinical concepts. Several types of concepts exist within **ricu** and with extensibility in mind, new types can easily be added. A quick remark on terminology before diving into more details on how to specify data concepts: A *concept* corresponds to a clinical variable such as a bilirubin measurement or the ventilation status of a patient, and an *item* encodes how to retrieve data corresponding to a given concept from a data source. A *concept* therefore contains several *items* (zero, one or multiple are possible per data source).

All concepts consist of minimal metadata including a name, target class (defaults to `ts_tbl`; see Section 4.1), an aggregation specification[6] and class information (`num_concept` if not otherwise specified), as well as optional `description` and `category` information. Adding to that, depending on concept class, further fields can be supplied. In the case of the most widespread concept type (`num_cncpt`; used to represent numeric data) this is `unit` which encodes one (or several synonymous) unit(s) of measurement, as well as a minimal and maximal plausible values (specified as `min` and `max`). The concept for heart rate data (`hr`) for example can be specified as

```
{
  "hr": {
    "unit": ["bpm", "/min"],
    "min": 0,
    "max": 300,
    "description": "heart rate",
    "category": "routine vital signs",
    "sources": {
      ...
    }
  }
}
```

[6]Every concept needs a default aggregation method which can be used during data loading to return data that is unique per key (either per `id_vars` group or per combination of `id_vars` and `index_var`) otherwise down-stream merging of multiple concepts is ill-defined. The aggregation default can be manually overridden during loading or automatically, by specification as part of a `rec_cncpt` object. If no aggregation method is explicitly indicated the global default is `first()` for character, `median()` for numeric and `any()` for logical vectors.

Metadata is used during concept loading for data-preprocessing. For numeric concepts, the specified measurement unit is compared to that of the data (if available), with messages being displayed in case of mismatches, while the range of plausible values is used to filter out measurements that fall outside the specified interval. Other types of concepts include categorical concepts (`fct_cncpt`), concepts representing binary data (`lgl_cncpt`), as well as recursive concepts (`rec_cncpt`), which build on other *atomic* concepts[7].

Finally, the most recently added concept class, `unt_cncpt`, inheriting from `num_cncpt`, aims to simplify manual conversion to target units, leveraging capabilities provided by the **units** package. For this to work, both source and target units have to be recognized and convertible (as reported by `units::ud_are_convertible()`). Measurement units that are not available by default can be registered using `units::install_unit()`.

Specification of how data can be retrieved from a data source is encoded by data *items*. Lists of data items (associated with data source names) are provided as `sources` element. For the demo datasets corresponding to eICU and MIMIC-III, heart rate data retrieval is specified as

```
{
  "eicu_demo": [
    {
      "table": "vitalperiodic",
      "val_var": "heartrate",
      "class": "col_itm"
    }
  ],
  "mimic_demo": [
    {
      "ids": [211, 220045],
      "table": "chartevents",
      "sub_var": "itemid"
    }
  ]
}
```

Analogously to how different concept classes are used to represent different types of data, different item classes handle different data loading requirements. The most common scenario is selecting a subset of rows from a table by matching a set of ID values (`sub_itm`). In the above example, heart rate data in MIMIC-III can be located by searching for ID values 211 and 220045 in column `itemid` of table `chartevents` (heart rate data is stored in *long* format). Conversely, heart rate data in eICU is stored in *wide* format, requiring no row-subsetting. Column `heartrate` of table `vitalperiodic` contains all corresponding data and such data situations are handled by the `col_itm` class. Other item classes include `rgx_itm` where a

[7]An example for a recursive concept is the PaO_2/FiO_2 ratio, used for instance to assess patients with acute respiratory distress syndrome (ARDS) or for Sepsis-Related Organ Failure Assessment (SOFA) (Villar, Pérez-Méndez, Blanco, Añón, Blanch, Belda, Santos-Bouza, Fernández, Kacmarek, and Spanish Initiative for Epidemiology and Therapies for ARDS (SIESTA) Network 2013; Vincent *et al.* 1996). Given both PaO_2 and FiO_2 as individual concepts, the PaO_2/FiO_2 ratio is provided by **ricu** as a recursive concept (`pafi`), requesting the two atomic concepts `pao2` and `fio2` and performing some form of imputation for when at a given time step one or both values are missing.

regular expression is used for selecting rows and `fun_itm` where an arbitrary function can be used for data loading. If a data loading scenario is not covered by these classes, adding further `itm` subclasses is encouraged.

In order to extend the current concept library both to new datasets and new concepts, further JSON files can be incorporated by adding paths to their enclosing directories to `RICU_CONFIG_PATH`. Concepts with names that exist in files of the same name but with higher precedence are only used for their `sources` entries, such that `hr` for `new_dataset` can be specified as follows, while concepts with non-existing names are treated as new concepts.

```
"hr": {
  "sources": {
    "new_dataset": [
      {
        "ids": 6640,
        "table": "numericitems",
        "sub_var": "itemid"
      }
    ]
  }
}
```

Central to providing the required flexibility for loading of certain data concepts that require some specific preprocessing are callback functions that can be specified for several *item* types. Functions (with appropriate signatures), designated as callback functions, are invoked on individual data items, before concept-related preprocessing is applied. A common scenario for this is unit of measurement conversion: In MIMIC-III data for example, several `itemid` values correspond to temperature measurements, some of which refer to temperatures measured in degrees Celsius whereas others are used for measurements in degrees Fahrenheit. As the information encoding which measurement corresponds to which `itemid` values is no longer available during concept-related preprocessing, this is best resolved at the level of individual data items. Several function factories are available for generating callback functions and `convert_unit()` is intended for covering unit conversions[8]. Data *items* corresponding to the `temp` concept for MIMIC-III are specified as

```
{
  "mimic_demo": [
    {
      "ids": [676, 677, 223762],
      "table": "chartevents",
      "sub_var": "itemid"
    },
    {
      "ids": [678, 679, 223761, 224027],
```

[8]The presented implementation of this concept predates the addition of automatic unit conversion using the **units** package. While the concept definition as used by **ricu** will be updated to reflect these new capabilities, this example remains for illustration purposes.

```
      "table": "chartevents",
      "sub_var": "itemid",
      "callback": "convert_unit(fahr_to_cels, 'C', 'f')"
    }
  ]
}
```

indicating that for ID values 676, 677 and 223762 no preprocessing is required and for the remaining ID values the function `fahr_to_cels()` is applied to entries of the `val_var` column where the regular expression `"f"` is `TRUE` for the `unit_var` column (the values of which being ultimately replaced with `"C"`).

5. Data sources

Every dataset is represented by an environment with class attributes and associated metadata objects stored as object attributes to that environment. Dataset environments all inherit from `src_env` and from any number of class names constructed from data source name(s) with a suffix `_env` attached. The environment representing MIMIC-III, for example inherits from `src_env` and `mimic_env`, while the corresponding demo dataset inherits from `src_env`, `mimic_env` and `mimic_demo_env`. These sub-classes are later used for tailoring the process of data loading to particularities of individual datasets.

A `src_env` contains an active binding per associated table, which returns a `src_tbl` object representing the requested table. As is the case for `src_env` objects, `src_tbl` objects inherit from additional classes for reasons explained above. The `admissions` table of the MIMIC-III demo dataset for example, inherits from `mimic_demo_tbl` and `mimic_tbl` (alongside classes `src_tbl` and `prt`).

```
R> mimic_demo$admissions

# <mimic_tbl>: [129 x 19]
# ID options:  subject_id (patient) < hadm_id (hadm) < icustay_id (icustay)
# Defaults:    `admission_type` (val)
# Time vars:   `admittime`, `dischtime`, `deathtime`, `edregtime`,
#   `edouttime`
    row_id subject_id hadm_id admittime           dischtime
     <int>      <int>   <int> <dttm>              <dttm>
  1  12258      10006  142345 2164-10-23 21:09:00 2164-11-01 17:15:00
  2  12263      10011  105331 2126-08-14 22:32:00 2126-08-28 18:59:00
  3  12265      10013  165520 2125-10-04 23:36:00 2125-10-07 15:13:00
  4  12269      10017  199207 2149-05-26 17:19:00 2149-06-03 18:42:00
  5  12270      10019  177759 2163-05-14 20:43:00 2163-05-15 12:00:00
...
125  41055      44083  198330 2112-05-28 15:45:00 2112-06-07 16:50:00
126  41070      44154  174245 2178-05-14 20:29:00 2178-05-15 09:45:00
127  41087      44212  163189 2123-11-24 14:14:00 2123-12-30 14:31:00
128  41090      44222  192189 2180-07-19 06:55:00 2180-07-20 13:00:00
```

```
129  41092      44228  103379 2170-12-15 03:14:00 2170-12-24 18:00:00
# ... with 119 more rows, and 14 more variables: deathtime <dttm>,
#   admission_type <chr>, admission_location <chr>,
#   discharge_location <chr>, insurance <chr>, language <chr>,
#   religion <chr>, marital_status <chr>, ethnicity <chr>,
#   edregtime <dttm>, edouttime <dttm>, diagnosis <chr>,
#   hospital_expire_flag <int>, has_chartevents_data <int>
```

Powered by the **prt** (Bennett 2021) package, `src_tbl` objects represent row-partitioned tabular data stored as multiple binary files created by the **fst** (Klik 2020) package. In addition to standard subsetting, `prt` objects can be subsetted via the base R S3 generic function `subset()` and using non-standard evaluation (NSE):

```
R> subset(mimic_demo$admissions, subject_id > 44000, language:ethnicity)
```

```
   language          religion marital_status              ethnicity
1:     ENGL          CATHOLIC         SINGLE                  WHITE
2:     ENGL          CATHOLIC         SINGLE                  WHITE
3:     ENGL          CATHOLIC         SINGLE                  WHITE
4:     ENGL PROTESTANT QUAKER        MARRIED                  WHITE
5:     ENGL      UNOBTAINABLE         SINGLE BLACK/AFRICAN AMERICAN
6:     ENGL          CATHOLIC         SINGLE                  WHITE
7:     ENGL     NOT SPECIFIED         SINGLE                  WHITE
```

This syntax makes it possible to read row-subsets of *long* tables into memory with little memory overhead. While terseness of such an API does introduce potential ambiguity, this is mostly overcome by using the tidy eval framework provided by **rlang** (Henry and Wickham 2020):

```
R> subject_id <- 44000:45000
R> subset(mimic_demo$admissions, .data$subject_id %in% .env$subject_id,
+        subject_id:dischtime)
```

```
   subject_id hadm_id           admittime           dischtime
1:      44083  125157 2112-05-04 08:00:00 2112-05-11 14:15:00
2:      44083  131048 2112-05-22 15:37:00 2112-05-25 13:30:00
3:      44083  198330 2112-05-28 15:45:00 2112-06-07 16:50:00
4:      44154  174245 2178-05-14 20:29:00 2178-05-15 09:45:00
5:      44212  163189 2123-11-24 14:14:00 2123-12-30 14:31:00
6:      44222  192189 2180-07-19 06:55:00 2180-07-20 13:00:00
7:      44228  103379 2170-12-15 03:14:00 2170-12-24 18:00:00
```

By using **rlang** pronouns (`.data` and `.env`), the distinction can readily be made between a name referring to an object within the context of the data and an object within the context of the calling environment.

5.1. Data source setup

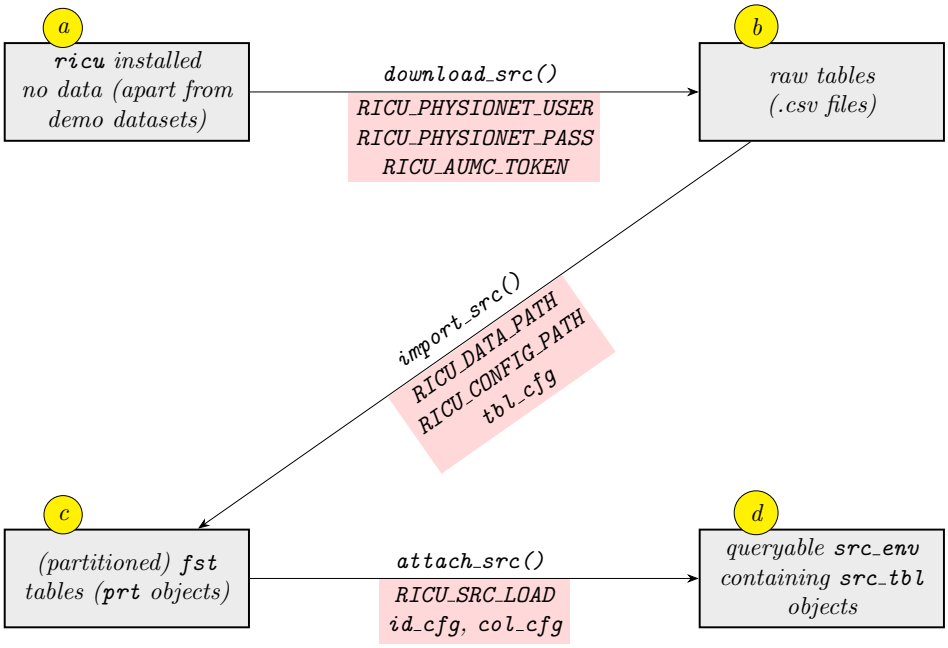


Figure 1: Making a dataset available to **ricu** involves several steps, starting with data download, followed by preparation for efficient access and finalized by instantiation of data structures containing relevant metadata. The functions which are used for each step are displayed above arrows and below (in red) are indicated specific configuration settings or environment variables which are need for (or can be used to customize) the specific step.

In order to make a dataset accessible to **ricu**, three steps are necessary, each handled by an exported S3 generic function: `download_scr()`, `import_src()` and `attach_src()`. The first two steps, data download and import, are one-time procedures, whereas attaching is carried out every time the package namespace is loaded. By default, all data sources known to **ricu** are configured to be attached and in case some data is missing for a given data source, the missing data is downloaded and imported on first access. An outline of the steps involved for data source setup is shown in Figure 1.

Data download

The first step towards accessing data is data download, taken care of by the S3 generic function `download_src()`. For the datasets included with **ricu**, prior to calling `download_src()`, the following environment variables can be set (indicated in red in the $a \to b$ edge in Figure 1):

- `RICU_PHYSIONET_USER`/`RICU_PHYSIONET_PASS`: PhysioNet login credentials with access to the requested dataset(s).
- `RICU_AUMC_TOKEN`: Download token, extracted from the download URL received after being granted data access.

If any of the required access credentials are not available as environment variables, they can be supplied as function arguments to `download_src()` or the user is queried in interactive sessions and an error is thrown otherwise.

As a quick reminder on system requirements for initial data setup operations: Each of the supported datasets requires 5-10 GB disk space for permanent storage and 50-100 GB of temporary disk storage during download and import. Memory requirements are kept low (8-16 GB) by performing all setup operations only on subsets of rows at the time. Initial data source setup can be expected to take upwards of an hour per dataset.

Data import

After successful data download, importing prepares tables for efficient random row- and column-access, for which the raw data format (.csv) is not well suited (see edge $b \rightarrow c$ in Figure 1). Tables are read in using **readr** (Wickham and Hester 2020), potentially (re-)partitioned row-wise, and re-saved using **fst**. Environment variables that can be set to customize **ricu** data handling, relevant for import and attaching include:

- `RICU_DATA_PATH`: Optional data storage location (if unset, this defaults to a system-specific, user-specific directory). The current value used for this setting can be queried by calling `data_dir()`.
- `RICU_CONFIG_PATH`: A comma-separated set of paths to directories containing configuration files. The current set of paths is retrievable by calling `config_paths()` and the ordering of paths determines precedence of how configuration files are combined (if multiple files of the same name are available).

For importing, the information contained in `tbl_cfg` configuration objects is most relevant. This determines column data types, table partitioning and sanity checks like number of rows per table. Please refer to Section 5.3.3 for more information on the construction of `tbl_cfg` objects.

Data attaching

Finally, attaching a dataset creates a corresponding `src_env` object, containing a corresponding `src_tbl` object for each table, which together with associated metadata are used by **ricu** to run queries against the data (edge $c \rightarrow d$ in Figure 1). The environment variable `RICU_SRC_LOAD` may contain a comma-separated list of data source names that are set up for being automatically attached on namespace loading. This defaults to all currently supported datasets and the active set of source names is available as `auto_attach_srcs()`. Apart from this automatism, the process of attaching a dataset can be manually invoked by calling `attach_src()`, which can be convenient when for example updating the data source configuration after it has been modified.

Two configuration objects which are important for data loading (see the following Section 5.2) are `id_cfg` and `col_cfg` (described in Sections 5.3.1 and 5.3.2, respectively), providing default values for certain types of columns, including time-stamp, measurement value and measurement unit column names, as well as defining relationships between patient identifiers (such as hospital stay ID and ICU stay ID).

5.2. Data loading

The lowest level of data access is direct subsetting of `src_tbl` objects as shown at the start of Section 5. As `src_tbl` inherits from `prt`, the `subset()` implementation provided by **prt** can

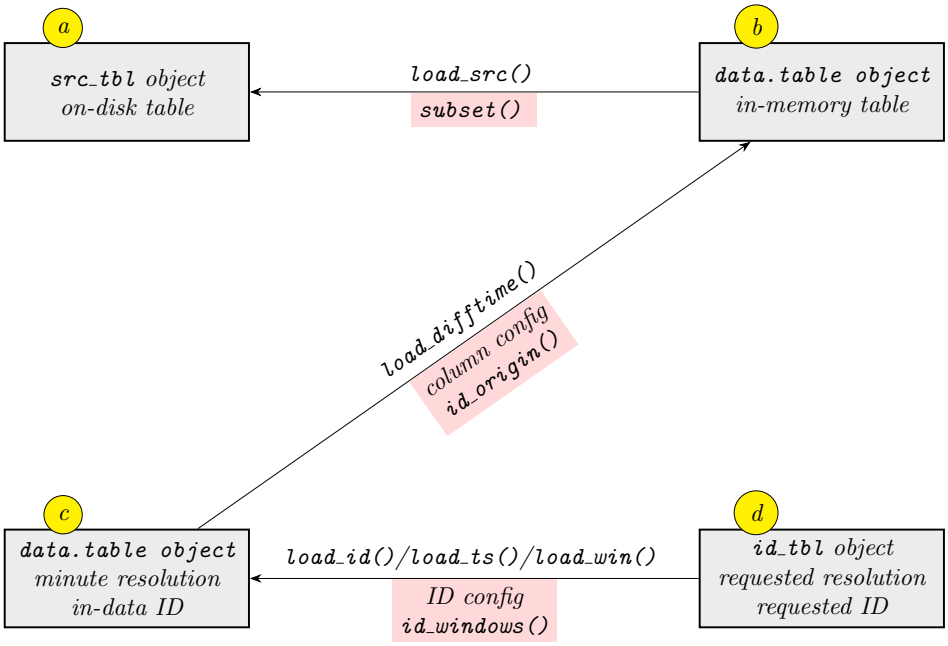


Figure 2: Data loading proceeds through several layers, each contributing a step towards harmonizing discrepancies among raw data representations provided by the different data sources. Raw data tables are represented by **ricu** as `src_tbl` objects which can be queried using `load_src()`. Absolute time-stamps in the returned `data.table` are converted to times relative to admission (in minutes) by `load_difftime()` and finally, `load_id()`/`load_ts()`/`load_win()` ensure a given ID system and time interval.

be used for NSE of data-expressions against on-disk, tabular data. Building on that, several S3 generic functions successively homogenize data representations as visualized in Figure 2.

The most basic layer in data loading is provided by the S3 generic function `load_src()`, which provides a string-based interface to the `cols` argument of `subset()` while forwarding the unevaluated expression passed as `rows` (see edge $a \rightarrow b$ in Figure 2).

```
R> load_src(mimic_demo$admissions, subject_id > 44000,
+           cols = c("hadm_id", "admittime", "dischtime"))

   hadm_id           admittime           dischtime
1:  125157 2112-05-04 08:00:00 2112-05-11 14:15:00
2:  131048 2112-05-22 15:37:00 2112-05-25 13:30:00
3:  198330 2112-05-28 15:45:00 2112-06-07 16:50:00
4:  174245 2178-05-14 20:29:00 2178-05-15 09:45:00
5:  163189 2123-11-24 14:14:00 2123-12-30 14:31:00
6:  192189 2180-07-19 06:55:00 2180-07-20 13:00:00
7:  103379 2170-12-15 03:14:00 2170-12-24 18:00:00
```

As data sources differ in their representation of time-stamps, a next step in data homogenization is to converge to a common format: the time difference to the origin time-point of a given ID system (for example ICU admission).

```
R> load_difftime(mimic_demo$admissions, subject_id > 44000,
+                cols = c("hadm_id", "admittime", "dischtime"))

# An ‘id_tbl‘: 7 x 3
# Id var:      ‘hadm_id‘
  hadm_id admittime dischtime
    <int> <drtn>    <drtn>
1  103379 0 mins    13846 mins
2  125157 0 mins    10455 mins
3  131048 0 mins     4193 mins
4  163189 0 mins    51857 mins
5  174245 0 mins      796 mins
6  192189 0 mins     1805 mins
7  198330 0 mins    14465 mins
```

The function `load_difftime()` is expected to return timestamps as base R `difftime` vectors (in minutes; edge $b \rightarrow c$ in Figure 2). The argument `id_hint` can be used to specify a preferred ID system, but if not available in raw data, `load_difftime()` will return data using the ID system with highest cardinality (i.e., ICU stay ID is preferred over hospital stay ID). In the above example, if `icustay_id` were requested, data would be returned using `hadm_id`, whereas a `subject_id` request would be honored, as the corresponding ID column is available in the `admissions` table.

Building on `load_difftime()` functionality, functions `load_id()`/`load_ts()`/`load_win()` return `id_tbl`/`ts_tbl`/`win_tbl` objects with the requested ID system (passed as `id_var` argument). This uses raw data IDs if available or calls `change_id()` in order to convert to the desired ID system (edge $c \rightarrow d$ in Figure 2). Similarly, where `load_difftime()` returns data with fixed time interval of one minute, `load_id()` allows for arbitrary time intervals (using `change_interval()`; defaults to 1 hour).

```
R> load_id(mimic_demo$admissions, subject_id > 44000,
+         cols = c("admittime", "dischtime"), id_var = "hadm_id")

# An ‘id_tbl‘: 7 x 3
# Id var:      ‘hadm_id‘
  hadm_id admittime dischtime
    <int> <drtn>    <drtn>
1  103379 0 hours   230 hours
2  125157 0 hours   174 hours
3  131048 0 hours    69 hours
4  163189 0 hours   864 hours
5  174245 0 hours    13 hours
6  192189 0 hours    30 hours
7  198330 0 hours   241 hours
```

Throughout several of theses functions, `col_cfg` objects are used to provide sensible defaults. In order to convert to relative times, `load_difftime()`, for example, requires names

of columns for which this applies (provided by the `time_vars` entry), and `load_ts()` needs to know which of the `time_vars` to use as `index_var`. For more information on the construction of `col_cfg` objects, please refer to Section 5.3.2.

A call to `change_id()` requires the construction of a table which contains the mapping between different ID systems, together with information about how to convert timestamps between these ID systems (edge $c \rightarrow d$ in Figure 2). The function responsible for providing the necessary information is `id_windows()` and the associated S3 generic function `id_win_helper()`. The entry point `id_windows()` wraps `id_win_helper()`, providing memoization, as the resulting structure is expensive to compute relative to the frequency of being required.

```
R> id_windows(mimic_demo)

# An ‘id_tbl‘: 136 x 9
# Id var:      ‘icustay_id‘
    icustay_id hadm_id subject_id icustay_id_start hadm_id_start
         <int>   <int>      <int> <drtn>           <drtn>
  1     201006  198503      10076 0 mins            -3290 mins
  2     201204  114648      42321 0 mins               -2 mins
  3     203766  126949      10045 0 mins            -1336 mins
  4     204132  157609      40310 0 mins               -1 mins
  5     204201  177678      10104 0 mins             -368 mins
...
132     295043  170883      10124 0 mins           -10413 mins
133     295741  176805      10090 0 mins               -1 mins
134     296804  110244      10035 0 mins            -1294 mins
135     297782  167612      43909 0 mins               -1 mins
136     298685  151323      42075 0 mins               -1 mins
# ... with 126 more rows, and 4 more variables: subject_id_start <drtn>,
#   icustay_id_end <drtn>, hadm_id_end <drtn>, subject_id_end <drtn>
```

Analogously, the function pair `id_origin()` and `id_orig_helper()`, with the former wrapping the latter and again providing memoization, is used for datasets where time-stamps are represented by absolute times, returning the origin time-points for a given ID system which then can be used to calculate relative times (edge $b \rightarrow c$ in Figure 2).

```
R> id_origin(mimic_demo, "icustay_id")

# An ‘id_tbl‘: 136 x 2
# Id var:      ‘icustay_id‘
    icustay_id intime
         <int> <dttm>
  1     201006 2107-03-24 04:06:14
  2     201204 2121-12-07 20:50:36
  3     203766 2129-11-24 22:46:57
  4     204132 2144-12-24 16:16:41
```

```
  5     204201 2120-08-24 23:47:23
...
132     295043 2192-04-24 02:29:49
133     295741 2124-01-12 14:27:16
134     296804 2129-03-04 13:40:11
135     297782 2152-10-09 19:05:36
136     298685 2166-02-12 17:57:37
# ... with 126 more rows
```

For the included datasets, the implementations of `id_win_helper()` and `id_orig_helper()`, use information contained in `id_cfg` objects (see Section 5.3.1) to determine which columns in which tables are required for constructing the corresponding lookup tables. Doing so, however, is not necessary: an `id_win_helper()` implementation for a new dataset could forego this by hard-coding table/column names as part of the function logic, in-turn simplifying the corresponding `id_cfg` object to merely providing naming and ordering information.

5.3. Data source configuration

Data source environments (and corresponding `src_tbl` objects) are constructed using source configuration objects: list-based structures, inheriting from `src_cfg` and from any number of data source specific class names with suffix `_cfg` appended (as discussed at the beginning of Section 5). The exported function `load_src_cfg()` reads a JSON formatted file and creates a `src_cfg` object per data source and further therein contained objects.

```
R> cfg <- load_src_cfg("mimic_demo")
R> str(cfg, max.level = 3L, width = 70L)

List of 1
 $ mimic_demo:List of 6
  ..$ name   : chr "mimic_demo"
  ..$ prefix : chr [1:2] "mimic_demo" "mimic"
  ..$ id_cfg : id_cfg [1:3] ‘subject_id‘, ‘hadm_id‘, ‘icustay_id‘
  ..$ col_cfg: col_cfg [1:25] [0, 0, 5, 0, 1], [0, 1, 6, 0, 1], [1, 0, 0...
  ..$ tbl_cfg: tbl_cfg [1:25] [?? x 19; 1], [?? x 24; 1], [?? x 4; 1], [...
  ..$ extra  :List of 1
  .. ..$ url: chr "https://physionet.org/files/mimiciii-demo/1.4"
  ..- attr(*, "class")= chr [1:3] "mimic_demo_cfg" "mimic_cfg" "src_cfg"

R> mi_cfg <- cfg[["mimic_demo"]]
```

In addition to required fields `name` and `prefix` (used as class prefix), as well as further arbitrary fields contained in `extra` (`url` in this case), several configuration objects are part of `src_cfg`: `id_cfg`, `col_cfg` and `tbl_cfg`.

ID configuration

An `id_cfg` object contains an ordered set of key-value pairs representing patient identifiers in a dataset. An implicit assumption currently is that a given patient ID system is used

consistently throughout a dataset, meaning that for example an ICU stay ID is always referred to by the same name throughout all tables containing a corresponding column. Owing to the relational origins of these datasets this has been fulfilled in all instances encountered so far. In MIMIC-III, ID systems

```
R> as_id_cfg(mi_cfg)
```

```
<id_cfg<mimic_demo[patient < hadm < icustay]>[3]>
     patient         hadm      icustay
‘subject_id‘    ‘hadm_id‘ ‘icustay_id‘
```

are available, allowing for identification of individual patients, their (potentially multiple) hospital admissions over the course of the years and their corresponding ICU admissions (as well as potential re-admissions). Ordering corresponds to cardinality: moving to larger values implies moving along a one-to-many relationship. This information is used in data-loading, whenever the target ID system is not contained in the raw data.

Default column configuration

Again used in data loading, this per-table set of key-value pairs specifies column defaults as `col_cfg` object. Each key describes a type of column with special meaning and the corresponding value specifies said column for a given table. The print method for `col_cfg` reports all keys alongside the per-table counts of accordingly registered values (i.e., columns).

```
R> as_col_cfg(mi_cfg)
```

```
<col_cfg<mimic_demo[id_var, index_var, time_vars, unit_var, val_var]>[25]>
        admissions            callout         caregivers        chartevents
   [0, 0, 5, 0, 1]    [0, 1, 6, 0, 1]    [1, 0, 0, 0, 1]    [0, 1, 2, 1, 1]
         cptevents              d_cpt    d_icd_diagnoses   d_icd_procedures
   [0, 1, 1, 0, 1]    [1, 0, 0, 0, 1]    [1, 0, 0, 0, 1]    [1, 0, 0, 0, 1]
          d_items         d_labitems     datetimeevents      diagnoses_icd
   [1, 0, 0, 0, 1]    [1, 0, 0, 0, 1]    [0, 1, 3, 0, 1]    [0, 0, 0, 0, 1]
         drgcodes           icustays     inputevents_cv     inputevents_mv
   [0, 0, 0, 0, 1]    [0, 1, 2, 0, 1]    [0, 1, 2, 1, 1]    [0, 1, 4, 1, 1]
       labevents microbiologyevents       outputevents           patients
   [0, 1, 1, 1, 1]    [0, 1, 2, 0, 1]    [0, 1, 2, 1, 1]    [0, 0, 4, 0, 1]
     prescriptions procedureevents_mv     procedures_icd           services
   [0, 1, 2, 1, 1]    [0, 1, 4, 1, 1]    [0, 0, 0, 0, 1]    [0, 1, 1, 0, 1]
         transfers
   [0, 1, 2, 0, 1]
```

The following column defaults are currently in use throughout **ricu** but the set of keys can be extended to arbitrary new values:

- `id_var`: In case a table does not contain at least one ID column corresponding to one of the ID systems specified as `id_cfg`, the default ID column can be set on a per-table

basis as `id_var`[9].

- `index_var`: A column that is used to define an ordering in time over rows, thereby providing a time series index[10].
- `time_vars`: Columns which will be treated as time variables (important for converting between ID systems for example), but not as time series indices[11].
- `unit_var`: Used in concept loading (more specifically for `num_cncpt` concepts, see Section 4.3) to identify columns that represent unit of measurement information.
- `val_var`: Again used when loading data concepts, this identified a default value variable in a table, representing the column of interest to be used as returned data column.

While `id_var`, `index_var` and `time_vars` are used to provide sensible defaults to functions used for general data loading (Section 5.2), `unit_var`, `val_var`, as well as potential user-defined defaults are only used in concept loading (see Section 4.2) and therefore need not be prioritized when integrating new data sources until data concepts have been mapped.

Table configuration

Finally, `tbl_cfg` objects are used during the initial setup of a data source. In order to create a representation of a table that is accessible by **ricu** from raw data, several key pieces of information are required:

- File name(s): In the simplest case, a single file corresponds to a single table. Other scenarios that have been encountered (and are therefore handled) include tables partitioned into multiple files and .tar archives containing multiple tables.
- Column specification: For each column, the expected data type has to be known, as well as a pair of names, one corresponding to the raw data column name and one corresponding to the column name to be used within **ricu**.
- (Optional) number of rows: Used as sanity check whenever available.
- (Optional) partitioning information: For very *long* tables it can be useful to specify a row-partitioning. This currently is only possible by applying a vector of breakpoints to a single numeric column, thereby defining a grouping.

Table configuration objects are only used within the context of the functions `download_src()` and `import_src()` and are therefore not required if download and import are carried out manually.

```
R> as_tbl_cfg(mi_cfg)
```

[9] This for example is the case for the `d_items` table in MIMIC-III, which does not contain any patient related data, but holds information on items encoding types of measurements, procedures, etc., used throughout other tables holding actual patient data.

[10] For the MIMIC-III table `inputevents_mv`, of the four available time variables (`starttime`, `endtime`, `storetime`, `comments_date`), `starttime` lends itself to be used as index variable more than the other candidates and therefore is set as default.

[11] In case of the `admissions` table in MIMIC-III for example, a total of five columns are considered to be time variables, none of which stands out as potential `index_var`.

```
<tbl_cfg<mimic_demo[rows x cols; partitions]>[25]>
        admissions            callout         caregivers        chartevents
      [?? x 19; 1]       [?? x 24; 1]        [?? x 4; 1]       [?? x 15; 2]
        cptevents              d_cpt    d_icd_diagnoses   d_icd_procedures
      [?? x 12; 1]        [?? x 9; 1]        [?? x 4; 1]        [?? x 4; 1]
         d_items         d_labitems     datetimeevents      diagnoses_icd
      [?? x 10; 1]        [?? x 6; 1]       [?? x 14; 1]        [?? x 5; 1]
         drgcodes           icustays     inputevents_cv     inputevents_mv
       [?? x 8; 1]       [?? x 12; 1]       [?? x 22; 1]       [?? x 31; 1]
        labevents microbiologyevents       outputevents           patients
       [?? x 9; 1]       [?? x 16; 1]       [?? x 13; 1]        [?? x 8; 1]
     prescriptions procedureevents_mv     procedures_icd           services
      [?? x 19; 1]       [?? x 25; 1]        [?? x 5; 1]        [?? x 6; 1]
        transfers
      [?? x 13; 1]
```

For the `chartevents` table of the MIMIC-III demo dataset, rows are partitioned into two groups, while all other tables are represented by a single partition. Furthermore, the expected number of rows is unknown (`??`) as this is missing from the corresponding `tbl_cfg` object.

5.4. Adding external datasets

In order to add a new dataset to **ricu**, several aspects outlined in the previous subsections require consideration. For illustration purposes, code for integrating AmsterdamUMCdb as external dataset is available from GitHub. While this is no longer needed for using the `aumc` data source, the repository will remain as it might serve as template to integration of new datasets. Throughout this repository (and the following paragraphs), the AmsterdamUMCdb data treated as an **ricu**-external dataset is referred to as `aumc_ext`.

Adding configuration information

Central to adding a new dataset to **ricu** is providing some configuration information in a `data-sources.json` file pointed to by the environment variable `RICU_CONFIG_PATH`. Depending on particularities of the dataset in question, corresponding implementations of some of the S3 generic functions mentioned throughout Sections 5.1 and 5.2 might have to be provided. The amount of confirmation information required to get started also depends on the desired level of integration. As data download and import are one-time procedures, these steps can be carried out manually, negating the need for specifying column data types in `data-sources.json` and providing data source specific methods for the `download_src()` and `import_src()` generics.

The basic organization of a data source configuration entry, as it could be used for `aumc_ext`, specified as JSON is as follows:

```
{
  "name": "aumc_ext",
  "id_cfg": {
    "patient": {
```

```
      "id": "patientid",
      "position": 1
    },
    "icustay": {
      "id": "admissionid",
      "position": 2
    }
  },
  "tables": {
    ...
  }
}
```

The shown `id_cfg` entry represents the minimally required set of entries, where for each ID specification, `start`, `end` and `table` are omitted (when compared to the `aumc` configuration provided by **ricu**). The `tables` entry expands to something like the following:

```
"tables": {
  "freetextitems": {
  },
  "drugitems": {
    "defaults": {
      "index_var": "start",
      "val_var": "dose",
      "unit_var": "doseunit",
      "time_vars": ["start", "stop"]
    }
  },
  "numericitems": {
    "defaults": {
      "index_var": "measuredat",
      "val_var": "value",
      "unit_var": "unit",
      "time_vars": ["measuredat", "registeredat", "updatedat"]
    },
    "partitioning": {
      "col": "",
      "breaks": [
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0
      ]
    }
  },
  ...
}
```

Minimally required is simply an entry indicating the data source membership of a table (if not partitioned; cf., `freetextitems`). This does slightly complicate data exploration, as if no

`defaults` are available, no default values can be provided to calls to `load_ts()` and related functions and therefore repeatedly have to be specified in corresponding function calls. Also, when specifying data items in such a setup, the per-table column names for special columns such as `index_var`, `val_var`, etc., have to be repeated for each individual item entry.

For partitioned tables, the basic structure of a `partitioning` entry is required, but the content itself is irrelevant, as this is only used for setup (cf., `numericitems`). The length of `breaks`, however, is required to match the number of partitions (i.e., a length 23 `breaks` specification corresponds to a partitioning into 24 row-groups.)[12]. The directory containing such a `data-sources.json` can then be pointed to by the environment variable `RICU_CONFIG_PATH`, making it available to **ricu**.

Enabling data loading

As for functions that are required, currently there is no default method available for the loading step provided by `load_difftime()` and most likely an implementation of the generic function `id_win_helper()` will be required as well. For `aumc_ext`, `load_difftime()` could be implemented as

```
R> ms_as_min <- function(x) {
+   as.difftime(as.integer(x / 6e4), units = "mins")
+ }
R>
R> aumc_difftime <- function(x, rows, cols = colnames(x),
+                            id_hint = id_vars(x),
+                            time_vars = ricu::time_vars(x), ...) {
+
+   if (id_hint %in% colnames(x)) {
+     id_sel <- id_hint
+   } else {
+     id_opt <- id_var_opts(sort(as_id_cfg(x), decreasing = TRUE))
+     id_sel <- intersect(id_opt, colnames(x))[1L]
+   }
+
+   stopifnot(is.character(id_sel), length(id_sel) == 1L)
+
+   if (!id_sel %in% cols) {
+     cols <- c(id_sel, cols)
+   }
+
+   time_vars <- intersect(time_vars, cols)
+
+   dat <- load_src(x, {{ rows }}, cols)
+   dat <- dat[, c(time_vars) := lapply(.SD, ms_as_min),
+              .SDcols = time_vars]
+
```

[12]Originally it was intended to use partitioning information during data loading in order to narrow down the set of partitions that have to be accessed. So far, this optimization has not been implemented.

```
+   as_id_tbl(dat, id_vars = id_sel, by_ref = TRUE)
+ }
```

Such a function attempts to use the ID as requested as `id_hint`, but falls back to the best possible alternative (using the ordering as previously specified in the `id_cfg` JSON configuration) if not provided by the data. The helper function `id_var_opts()` returns the dataset-specific column names of an `id_cfg` object (as opposed to the dataset-agnostic ID names; cf., `subject_id` and `patient`). Both the row-subsetting expression and column selection are passed on to `load_src()` and all columns specified as `time_vars` are converted to `difftime` vectors in minutes. Operations can safely be carried out using by-reference semantics, as intermediate objects are not exposed to the user.

For a possible implementation of the `id_win_helper()` generic, column and table names to assemble the desired lookup table are hard coded instead of provided by the corresponding `id_cfg` object (as is the case in the **ricu**-internal implementation).

```
R> aumc_windows <- function(x) {
+
+   ids <- c("admissionid", "patientid")
+   sta <- c("admittedat", "firstadmittedat")
+   end <- c("dischargedat", "dateofdeath")
+
+   tbl <- as_src_tbl(x, "admissions")
+
+   res <- tbl[, c(ids, sta[1L], end)]
+   res <- res[, c(sta[2L]) := 0L]
+   res <- res[, c(sta, end) := lapply(.SD, ms_as_min),
+              .SDcols = c(sta, end)]
+
+   res <- data.table::setcolorder(res, c(ids, sta, end))
+   res <- rename_cols(res, c(ids, paste0(ids, "_start"),
+                                  paste0(ids, "_end")), by_ref = TRUE)
+
+   as_id_tbl(res, ids[2L], by_ref = TRUE)
+ }
```

As all the required information is available form the `admissions` table, `aumc_windows()` simply loads the corresponding columns, converts them to minute resolution, followed by some renaming. ICU admissions and discharges in this table are relative to initial hospital admissions and therefore an all-zero column `firstadmittedat` is added and the `id_var` of the resulting `id_tbl` is marked as `patientid`[13].

A final step in making a new dataset accessible to **ricu** lies in specifying concept items. To this end, a file `concept-dict.json` can be added to the directory pointed to by the environment

[13]The patient ID created in this way is different to that available for MIMIC-III, where patient date of birth is provided. An approximate date of birth could be constructed if ages were reported more precisely, but given the rough binning available here, this might be considered an acceptable limitation of resulting patient IDs. Nevertheless awareness of such differences in data presentation is important.

variable `RICU_CONFIG_PATH`, containing entries like the following, which will make it possible to use the `hr` concept across all datasets included with **ricu**, alongside the newly added dataset.

```
{
  "hr": {
    "sources": {
      "aumc_ext": [
        {
          "ids": 6640,
          "table": "numericitems",
          "sub_var": "itemid"
        }
      ]
    }
  }
}
```

The above outline serves as an example on how to proceed when adding new data to **ricu**. Aspects like having multiple patient IDs, for example, could be further simplified[14]. Owing to the extensive use of S3 generic functions, **ricu** offers considerable flexibility for customizing certain behavior to specifics of a given data source, while providing fallback procedures whenever more general treatment can be applied.

Summary of required steps

Summarizing aspects explained in more detail in the previous sections, the following points list the required steps for adding new data in the order they should be considered in. The approach taken here being is to start simple and expand.

1. Tables saved as `.fst` files should be moved to the folder returned by `src_data_dir()` when passed the dataset name (alternatively, methods implementing `src_download()` and `src_import()` are required).

2. A minimal data source configuration file `data-sources.json` is required in the directory pointed to by `RICU_CONFIG_PATH`. For AmsterdamUMCdb, this could be as minimal as (assuming no partitioning):

```
{
  "name": "aumc_min",
  "id_cfg": {
    "icustay": "admissionid"
  },
  "tables": {
    "admissions": {},
```

[14] An example for such a reduced setup is available from the AUMC GitHub repository as `aumc_min`. Moving to only a single patient identifier also does away with the need for a `id_win_helper()` implementation, as `change_id()` will not be called in such a scenario.

```
    "drugitems": {},
    "freetextitems": {},
    "listitems": {},
    "numericitems": {},
    "procedureorderitems": {},
    "processitems": {}
  }
}
```

File names have to match table names, i.e., the admissions table should be named `admissions.fst`. Upon a call to `attach_src()` (or next loading of the package and having added the data source name to `RICU_SRC_LOAD`) the new data source can be explored using `load_src()`.

3. A `load_difftime()` method is required, which:

   - passes a row-subsetting expression to `load_src()` using the **rlang** curly-curly operator,
   - converts columns passed as `time_vars` to minute-resolution `difftime` vectors,
   - returns an `id_tbl` object where patient identifiers are chosen such that time-stamps are relative to corresponding admission,
   - (optionally) uses the column passed as `id_hint` for patient identifiers, if multiple identifiers are available from data.

   Upon registering this method with S3 dispatch, higher-level data loading functions such as `load_ts()` become available (given that no changes in patient identifiers are requested).

4. (Optional) if the source configuration specifies multiple patient identifiers which are not all available from all tables directly, an implementation of `id_win_helper()` most likely will be required (see Section 5.2).

5. Now, the source configuration can be expanded with per-table column defaults and data items can be added to the concepts included with **ricu** by creating a `concept-dict.json` under the path pointed to by `RICU_CONFIG_PATH`. For more information on readily available concepts, refer to Section 4.2 and for specifying new concepts altogether, pointers are available in section 4.3.

6. Examples

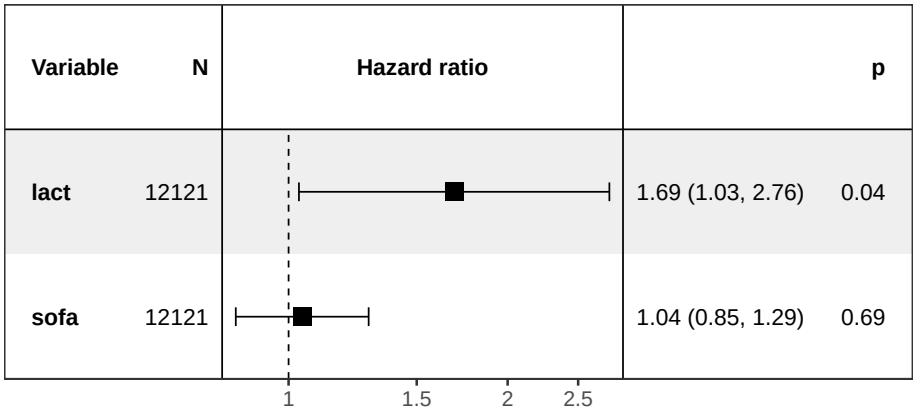
In order to briefly illustrate how **ricu** could be applied to real-world clinical questions, two examples are provided in the following sections. The first example fully relies on data concepts that are included with **ricu**. Whereas the second one explores both how some data preprocessing can be added to an existing concept, by creating a recursive concept (or `rec_cncpt`), as well as how to create an entirely new data concept in code (instead of JSON specification as outlined in Section 4.3), using constructors `item()` and `concept()`.

6.1. Lactate and mortality

First, the association of lactate levels and mortality is investigated. This problem has been studied before and it is widely accepted that both static and dynamic lactate indices are associated with increased mortality (Haas, Lange, Saugel, Petzoldt, Fuhrmann, Metschke, and Kluge 2016; Nichol, Bailey, Egi, Pettila, French, Stachowski, Reade, Cooper, and Bellomo 2011; Van Beest, Brander, Jansen, Rommes, Kuiper, and Spronk 2013). In order to model this relationship, a time-varying proportional hazards Cox model (Therneau and Grambsch 2000; Therneau 2021) is fitted, which includes the SOFA score as a general predictor of illness severity, using MIMIC-III demo data. Furthermore, for the sake of this example, the patient cohort is defined to be patients admitted from 2008 onwards (corresponding to the MetaVision database) of ages 20 to 90 years old.

```
R> src <- "mimic_demo"
R>
R> cohort <- load_id("icustays", src, dbsource == "metavision",
+                    cols = NULL)
R> cohort <- load_concepts("age", src, patient_ids = cohort,
+                          verbose = FALSE)
R>
R> dat <- load_concepts(c("lact", "death", "sofa"), src,
+                       patient_ids = cohort[age > 20 & age < 90, ],
+                       verbose = FALSE)
R>
R> dat <- dat[,
+   head(.SD, n = match(TRUE, death, .N)), by = c(id_vars(dat))
+ ]
R>
R> dat <- fill_gaps(dat)
R>
R> dat <- replace_na(dat, c(NA, FALSE), type = c("locf", "const"),
+                    by_ref = TRUE, vars = c("lact", "death"),
+                    by = id_vars(dat))
R>
R> cox_mod <- coxph(
+   Surv(charttime - 1L, charttime, death) ~ lact + sofa,
+   data = dat
+ )
```

After loading the data, some minor preprocessing is still required before modeling: first, data is filtered such that only data up to (and including) the hour in which the `death` flag switches to `TRUE` is used. Following that, missing values for `lact` are imputed using a last observation carry forward (LOCF) scheme (observing the patient grouping) and missing `death` values are set to `FALSE`. The resulting model fit can be visualized as:



A simple exploration already shows that the increased values of lactate are associated with mortality, even after adjusting for the SOFA score. Using abstractions provided by **ricu**, this analysis could now also be applied to other datasets with minimal effort.

6.2. Diabetes and insulin treatment

For the next example, again using MIMIC-III demo data, comorbidities and treatment related information are used: the amount of insulin administered to patients in the first 24 hours from their ICU admission is analyzed, in connection with diabetic status, in order to determine whether diabetic patients receive more insulin over that time-span, when compared to non-diabetic patients. For this, two concepts are introduced: `ins24`, a binned variable representing the cumulative amount of insulin administered within the first 24 hours of an ICU admission, and `diab`, a logical variable encoding diabetes comorbidity.

As there already is an insulin concept available, `ins24` can be implemented as `rec_cncpt`, requesting data from the `ins` concept. In order to be able to calculate the total amount of insulin administered, it is required to change the default aggregation method from `median()` to `sum()`. Failing to do so would yield under-reported values whenever several insulin administrations fall within a given time-step. The callback function `ins_cb()` is then inserted into the loading process, performing of the preprocessing steps outlined above: first data is subsetted to fall into the first 24 hours of ICU admissions, followed by binning of summed values.

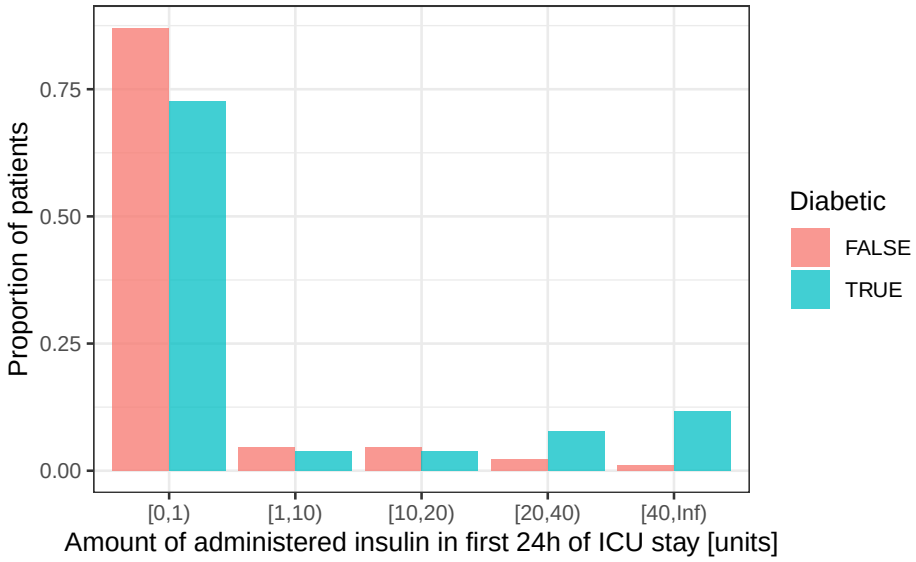
```
R> ins_breaks <- c(0, 1, 10, 20, 40, Inf)
R>
R> ins_cb <- function(ins, ...) {
+
+   day_one <- function(x) x >= hours(0L) & x <= hours(24L)
+
+   idx_var <- index_var(ins)
+   ids_var <- id_vars(ins)
+
+   ins <- ins[
+     day_one(get(idx_var)), list(ins24 = sum(ins)), by = c(ids_var)
+   ]
```

```
+
+   ins <- ins[,
+     ins24 := list(cut(ins24, breaks = ins_breaks, right = FALSE))
+   ]
+
+   ins
+ }
R>
R> ins24 <- load_dictionary(src, "ins")
R> ins24 <- concept("ins24", ins24, "insulin in first 24h",
+                   aggregate = "sum", callback = ins_cb,
+                   target = "id_tbl", class = "rec_cncpt")
```

The binary diabetes concept can be implemented as `lgl_cncpt`, for which ICD-9 codes are matched using a regular expression. As not only the subset of diabetic patients is of interest, a `col_itm` is more suited for diabetes status retrieval over a `rgx_itm`. For creating the required callback function, which produces a logical vector, the exported function factory `transform_fun()` can be employed, coupled with a function like `grep_diab()`, performing the desired transformation. The two concepts are then combined using `c()` and loaded via `load_concepts()`.

```
R> grep_diab <- function(x) {
+   grepl("^250\\.?[0-9]{2}$", x)
+ }
R>
R> diab  <- item(src, table = "diagnoses_icd",
+                callback = transform_fun(grep_diab),
+                class = "col_itm")
R>
R> diab  <- concept("diab", diab, "diabetes", target = "id_tbl",
+                   class = "lgl_cncpt")
R>
R> dat <- load_concepts(c(ins24, diab), id_type = "icustay",
+                        verbose = FALSE)
R> dat <- replace_na(dat, "[0,1)", vars = "ins24")
R>
R> dat
```

Following this, the difference between the two groups can be visualized with a histogram over the binned insulin administration values:



The plot suggests that for the MetaVision cohort defined in the previous example (without age subsetting) and during the first day of ICU stay, perhaps unsurprisingly, with increasing insulin dosage, diabetic patients receive more insulin compared to non-diabetic patients. This effect is more pronounced when looking at the full MIMIC-III data instead of the demo subset which includes only data corresponding to roughly 130 ICU stays.

7. Acknowledgments

Nicolas Bennett, Drago Plečko, Nicolai Meinshausen and Peter Bühlmann were supported by grant #2017-110 of the Strategic Focal Area "Personalized Health and Related Technologies (PHRT)" of the ETH Domain for the SPHN/PHRT Driver Project "Personalized Swiss Sepsis Study".

References

Adibuzzaman M, Musselman K, Johnson A, Brown P, Pitluk Z, Grama A (2016). "Closing the Data Loop: An Integrated Open Access Analysis Platform for the MIMIC Database." In *2016 Computing in Cardiology Conference (CinC)*, pp. 137–140. Institute of Electrical and Electronics Engineers.

Bennett N (2021). ***prt**: Tabular Data Backed by Partitioned 'fst' Files*. R package version 0.1.3, URL https://CRAN.R-project.org/package=prt.

Bone RC, Sibbald WJ, Sprung CL (1992). "The ACCP-SCCM Consensus Conference on Sepsis and Organ Failure." *Chest*, **101**(6), 1481–1483.

Desautels T, Calvert J, Hoffman J, Jay M, Kerem Y, Shieh L, Shimabukuro D, Chettipally U, Feldman MD, Barton C, *et al.* (2016). "Prediction of Sepsis in the Intensive Care Unit with Minimal Electronic Health Record Data: A Machine Learning Approach." *JMIR Medical Informatics*, **4**(3), e28.

Evans RS (2016). "Electronic Health Records: Then, Now, and in the Future." *Yearbook of Medical Informatics*, **25**(S 01), 48–61.

Faltys M, Zimmermann M, Lyu X, Hüser M, Hyland SL, Rätsch G, Merz TM (2021). "HiRID, A High Time-Resolution ICU Dataset (Version 1.1.1)." PhysioNet.

Fleuren LM, Klausch TLT, Zwager CL, Schoonmade LJ, Guo T, Roggeveen LF, Swart EL, Girbes ARJ, Thoral P, Ercole A, Hoogendoorn M, Elbers PWG (2020). "Machine Learning for the Prediction of Sepsis: A Systematic Review and Meta-Analysis of Diagnostic Test Accuracy." *Intensive Care Medicine*, **46**(3), 383–400.

Futoma J, Hariharan S, Sendak M, Brajer N, Clement M, Bedoya A, O'Brien C, Heller K (2017). "An Improved Multi-Output Gaussian Process RNN With Real-Time Validation for Early Sepsis Detection." ArXiv:1708.05894.

Goldberger AL, Amaral LAN, Glass L, Hausdorff JM, Ivanov PC, Mark RG, Mietus JE, Moody GB, Peng CK, Stanley HE (2000). "PhysioBank, PhysioToolkit and PhysioNet." *Circulation*, **101**(23), e215–e220.

Haas SA, Lange T, Saugel B, Petzoldt M, Fuhrmann V, Metschke M, Kluge S (2016). "Severe Hyperlactatemia, Lactate Clearance and Mortality in Unselected Critically Ill Patients." *Intensive Care Medicine*, **42**(2), 202–210.

Henry L, Wickham H (2020). ***rlang**: Functions for Base Types and Core R and 'Tidyverse' Features*. R package version 0.4.9, URL https://CRAN.R-project.org/package=rlang.

Hyland SL, Faltys M, Hüser M, Lyu X, Gumbsch T, Esteban C, Bock C, Horn M, Moor M, Rieck B, Zimmermann M, Bodenham D, Borgwardt K, Rätsch G, Merz TM (2020). "Early Prediction of Circulatory Failure in the Intensive Care Unit Using Machine Learning." *Nature Medicine*, **26**(3), 364–373.

Jiang F, Jiang Y, Zhi H, Dong Y, Li H, Ma S, Wang Y, Dong Q, Shen H, Wang Y (2017). "Artificial Intelligence in Healthcare: Past, Present and Future." *Stroke and Vascular Neurology*, **2**(4), 230–243.

Johnson A, Bulgarelli L, Pollard T, Horng S, Celi LA, Mark R (2021). "MIMIC-IV (Version 1.0)." PhysioNet.

Johnson AE, Pollard TJ, Shen L, Li-wei HL, Feng M, Ghassemi M, Moody B, Szolovits P, Celi LA, Mark RG (2016). "MIMIC-III, A Freely Accessible Critical Care Database." *Scientific Data*, **3**, 160035.

Johnson AEW, Aboab J, Raffa JD, Pollard TJ, Deliberato RO, Celi LA, Stone DJ (2018). "A Comparative Analysis of Sepsis Identification Methods in an Electronic Database." *Critical Care Medicine*, **46**(4), 494–499.

Jones M (2012). "NEWSDIG: The National Early Warning Score Development and Implementation Group." *Clinical Medicine*, **12**(6), 501–503. doi:10.7861/clinmedicine.12-6-501.

Kam HJ, Kim HY (2017). "Learning Representations for the Early Detection of Sepsis With Deep Neural Networks." *Computers in Biology and Medicine*, **89**, 248–255.

Klik M (2020). ***fst**: Lightning Fast Serialization of Data Frames*. R package version 0.9.4, URL https://CRAN.R-project.org/package=fst.

Lee J, Scott D, Villarroel M, Clifford G, Saeed M, Mark R (2011). "Open-access MIMIC-II Database for Intensive Care Research." In *Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, volume 2011, pp. 8315–8. Institute of Electrical and Electronics Engineers.

Moody GB, Mark RG (1996). "A Database to Support Development and Evaluation of Intelligent Intensive Care Monitoring." In *Computers in Cardiology*, pp. 657–660. Institute of Electrical and Electronics Engineers.

Nemati S, Holder A, Razmi F, Stanley MD, Clifford GD, Buchman TG (2018). "An Interpretable Machine Learning Model for Accurate Prediction of Sepsis in the ICU." *Critical Care Medicine*, **46**(4), 547–553.

Nichol A, Bailey M, Egi M, Pettila V, French C, Stachowski E, Reade MC, Cooper DJ, Bellomo R (2011). "Dynamic Lactate Indices as Predictors of Outcome in Critically Ill Patients." *Critical Care*, **15**(5), R242.

Ooms J (2014). "The **jsonlite** Package: A Practical and Consistent Mapping Between JSON Data and R Objects." ArXiv:1403.2805.

Pebesma E, Mailund T, Hiebert J (2016). "Measurement Units in R." *R Journal*, **8**(2), 486–494. doi:10.32614/RJ-2016-061.

Pollard TJ, Johnson AE, Raffa JD, Celi LA, Mark RG, Badawi O (2018). "The eICU Collaborative Research Database, A Freely Available Multi-Center Database for Critical Care Research." *Scientific Data*, **5**, 180178.

Singer M, Deutschman CS, Seymour CW, Shankar-Hari M, Annane D, Bauer M, Bellomo R, Bernard GR, Chiche JD, Coopersmith CM, Hotchkiss RS, Levy MM, Marshall JC, Martin GS, Opal SM, Rubenfeld GD, van der Poll T, Vincent JL, Angus DC (2016). "The Third International Consensus Definitions for Sepsis and Septic Shock (Sepsis-3)." *JAMA*, **315**(8), 801–810.

Subbe C, Kruger M, Rutherford P, Gemmel L (2001). "Validation of a Modified Early Warning Score in Medical Admissions." *QJM: An International Journal of Medicine*, **94**(10), 521–526.

Therneau TM (2021). *A Package for Survival Analysis in R*. R package version 3.2-11, URL https://CRAN.R-project.org/package=survival.

Therneau TM, Grambsch PM (2000). *Modeling Survival Data: Extending the Cox Model*. Springer-Verlag, New York.

Thoral PJ, Peppink JM, Driessen RH, Sijbrands EJG, Kompanje EJO, Kaplan L, Bailey H, Kesecioglu J, Cecconi M, Churpek M, Clermont G, van der Schaar M, Ercole A, Girbes ARJ, Elbers PWG, on behalf of the Amsterdam University Medical Centers Database (AmsterdamUMCdb) Collaborators, the SCCM/ESICM Joint Data Science Task Force (2021). "Sharing ICU Patient Data Responsibly Under the Society of Critical Care Medicine/European

Society of Intensive Care Medicine Joint Data Science Collaboration: The Amsterdam University Medical Centers Database (AmsterdamUMCdb) Example." *Critical Care Medicine*, **Latest Articles**.

Van Beest PA, Brander L, Jansen SP, Rommes JH, Kuiper MA, Spronk PE (2013). "Cumulative Lactate and Hospital Mortality in ICU Patients." *Annals of Intensive Care*, **3**(1), 6.

Villar J, Pérez-Méndez L, Blanco J, Añón JM, Blanch L, Belda J, Santos-Bouza A, Fernández RL, Kacmarek RM, Spanish Initiative for Epidemiology and Therapies for ARDS (SIESTA) Network S (2013). "A Universal Definition of ARDS: The PaO2/FiO2 Ratio Under a Standard Ventilatory Setting — A Prospective, Multicenter Validation Study." *Intensive Care Medicine*, **39**(4), 583–592.

Vincent JL, Moreno R, Takala J, Willatts S, De Mendonça A, Bruining H, Reinhart C, Suter P, Thijs LG (1996). "The SOFA (Sepsis-Related Organ Failure Assessment) Score to Describe Organ Dysfunction/Failure."

Wang RZ, Sun CH, Schroeder PH, Ameko MK, Moore CC, Barnes LE (2018). "Predictive Models of Sepsis in Adult ICU Patients." In *2018 IEEE International Conference on Healthcare Informatics (ICHI)*, pp. 390–391. Institute of Electrical and Electronics Engineers.

Wang S, McDermott MB, Chauhan G, Ghassemi M, Hughes MC, Naumann T (2020). "MIMIC-Extract: A Data Extraction, Preprocessing, and Representation Pipeline for MIMIC-III." In *Proceedings of the ACM Conference on Health, Inference, and Learning*, pp. 222–235. Association for Computing Machinery.

Wickham H, Hester J (2020). ***readr**: Read Rectangular Text Data*. R package version 1.4.0, URL https://CRAN.R-project.org/package=readr.

Wong A, Otles E, Donnelly JP, Krumm A, McCullough J, DeTroyer-Cooley O, Pestrue J, Phillips M, Konye J, Penoza C, Ghous M, Singh K (2021). "External Validation of a Widely Implemented Proprietary Sepsis Prediction Model in Hospitalized Patients." *JAMA Internal Medicine*.

Affiliation:

Nicolas Bennett[1]
ETH Zürich
Seminar for Statistics
Rämistrasse 101
CH-8092 Zurich
E-mail: nicolas.bennett@stat.math.ethz.ch

Drago Plečko[1]
ETH Zürich
Seminar for Statistics
Rämistrasse 101
CH-8092 Zürich
E-mail: drago.plecko@stat.math.ethz.ch

Ida-Fong Ukor
Monash Health
Department of Anaesthesiology and Perioperative Medicine
246 Clayton Road
Clayton VIC 3168
E-mail: ida-fong.ukor@monashhealth.org

Nicolai Meinshausen
ETH Zürich
Seminar for Statistics
Rämistrasse 101
CH-8092 Zürich
E-mail: meinshausen@stat.math.ethz.ch

Peter Bühlmann
ETH Zürich
Seminar for Statistics
Rämistrasse 101
CH-8092 Zürich
E-mail: peter.buehlmann@stat.math.ethz.ch

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd

[1]These authors contributed equally.