The Journal of Open Source Software

PatientProfiles: An R package to identify patient characteristics in data mapped to the OMOP common data model

Martí Català [1¶], **Mike Du** [1], **Yuchen Guo** [1], **Kim Lopez-Guell** [1], **Núria Mercadé-Besora** [1], **Xihang Chen** [1], **Marta Alcalde-Herraiz** [1], **Xintong Li** [1], **Daniel Prieto-Alhambra** [1,2], **and Edward Burn** [1]

1 Health Data Sciences Group, Nuffield Department of Orthopaedics, Rheumatology and Musculoskeletal Sciences, University of Oxford, United Kingdom **2** Department of Medical Informatics, Erasmus University Medical Center, Rotterdam, The Netherlands ¶ Corresponding author

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Nick Golding

Reviewers:

- @javier-gracia-tabuenca-tuni
- @pacoramon

Submitted: 09 July 2025
Published: unpublished

License
Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License (CC BY 4.0).

Summary

Real-world data (RWD) mapped to the Observational Medical Outcomes Partnership Common Data Model (OMOP CDM) offers a standardised framework for conducting observational health research across diverse data sources. However, identifying and summarising patient-level characteristics within this model often requires custom code, limiting efficiency and reproducibility. To address this, we developed the open-source PatientProfiles R package. This package streamlines the process of extracting demographic characteristics, computing intersections between cohorts and clinical events, and generating standard summaries of patient populations in OMOP CDM datasets.

Built on the tidyverse and omopgenerics infrastructure, PatientProfiles supports SQL translation for scalable database operations and includes comprehensive test coverage across multiple database systems. It provides a suite of functions grouped into demographics, intersections, summaries, utility, and mock data generation. The package is designed for transparency, modularity, and reusability in epidemiological workflows and is available via CRAN and GitHub, along with documentation and vignettes to support users.

Statement of need

Real-world data (RWD), routinely collected health data such as GP records, hospital data, and insurance claims data are valuable resources for conducting epidemiological research studies. However, with such data typically not collected primarily for research, different RWD sources can vary substantially in format and clinical coding systems. To overcome this difficulty a common data model (CDM) is often used. A CDM helps standardising data structures across various sources, enhancing data consistency, quality, and interoperability. A particularly popular data model is the Observational Medical Outcomes Partnership (OMOP) CDM, with more than 800 million patients' health care data transformed into this format (Overhage et al., 2011).

The OMOP CDM is a person-centric relational data model. Patients' data is spread across various tables related to different clinical domains with, for example, the *condition occurrence* table containing diagnoses while the *drug exposure* table contains drug prescriptions. These different clinical tables are all linked back to the *person* table which contains a unique identifier for each individual along with some key demographic data such as their date of birth.

DRAFT

The Journal of Open Source Software

Meanwhile, records in the *observation period* table define the period of calendar time over which an individual is followed-up.(Blacketer, 2025)

One of the principal benefits of mapping data to a CDM is that it allows for the same analytic code to be run across different datasets. Developing well-tested and easy to use software for common analytic tasks can therefore bring significant benefits, both improving the speed in which analyses can be performed and improved quality by reducing the amount of study-specific bespoke code needing to be written.

Obtaining the characteristics of individuals is one of the most common first tasks when working with patient-level data. In almost all analyses specific characteristics of individuals will need to be identified, after which groups of individuals who share some specific common condition or characteristic need to be identified and relationships between these groups are described (for example the time between a given diagnosis and a health outcome of interest).

We created the PatientProfiles R package to support identifying patient characteristics in data mapped to the OMOP CDM. It provides functionality to obtain demographic information (such as age, sex, prior observation time, future observation time, and so on), describe intersections between different groups of patients, and summarise the results in a standard output format.

Design principles

PatientProfiles was designed to adhere to the tidyverse tidy design principles. The tidyverse is a collection of R packages designed for data science, offering a cohesive and consistent syntax for data manipulation, and analysis (Wickham et al., 2019). The dplyr package defines multiple methods that can be implemented to many different sources of data. Of particular relevance to working with OMOP CDM data which is typically stored in a database, the dbplyr package provides translations of dplyr methods to SQL.

The core dependency of PatientProfiles is the omopgenerics package (Català & Burn, 2024), which provides methods, classes and basic operations for packages working with data in the OMOP CDM format. It defines a central object, a `cdm_reference`, that provides a central reference to all the different OMOP CDM tables, along with various other S3 classes and methods that facilitate working with the data contained in this reference.

Development of the PatientProfiles R package

PatientProfiles was developed in accordance with best practices for R packages with the devtools and usethis R packages used for common development tasks. The core, general dependencies of the package include dplyr and tidyr for common data manipulations and dbplyr which provides translations to SQL. In addition the core dependency related to OMOP CDM data is the omopgenerics package which provides core classes and methods specific to this data format.

The PatientProfiles package includes functionality to create its own mock data in the OMOP CDM format. This mock data is used to test the package using the testthat framework (Wickham, 2011). Every line of the packages is tested multiple times trying to account for various edge cases. Currently, the package is tested iteratively against different database management systems: PostgreSQL, SQL Server, Amazon Redshift, and DuckDB. In addition to unit tests, end-to-end integration tests of the package have been conducted to ensure the face validity of results.

The package is open-source and released via CRAN: https://CRAN.R-project.org/package=PatientProfiles (Català et al., 2025) (version 1.4.1 as of 9th July 2025) and also available on github: https://github.com/darwin-eu/PatientProfiles with its own website with more documentation and vignettes that cover the content of the package more in depth.

DRAFT

The Journal of Open Source Software

Overview of the PatientProfiles R package

PatientProfiles contains three main groups of functions (Figure 1). **Demographics** functions are used to add information contained in person and observation period tables to other tables or objects of interest. **Intersections** are used to intersect a table with an object of interest (it can be another table, a cohort of patients or a paritciar clinical concept). The **summarise** functions are used to create standard objects that summarise the content of a table of interest. Finally, the package also contains some complementary utility functions to for example create mock data.

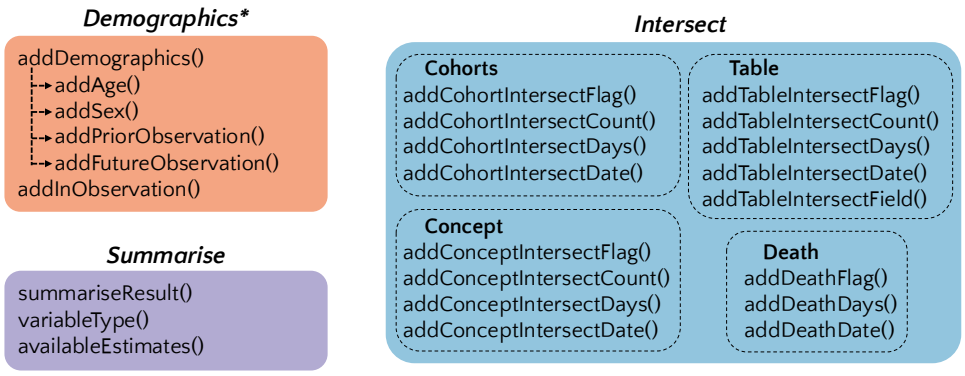

Figure 1: PatientProfiles functions bloks. Note that each demographic function has its own analogous *query* function to only add a query to the data, e.g. `addAge()` -> `addAgeQuery()`.

Mock data

A reference to an OMOP CDM instance is needed to use PatientProfiles. In this simple tutorial we will use mock toy data produced by the same package. By default this toy data is copied into an in-process duckdb database.

```
library(PatientProfiles)
cdm <- mockPatientProfiles(numberIndividuals = 1000)
# to customise cohorts
cdm$my_flu_cohort <- cdm$cohort1 |>
  dplyr::filter(cohort_definition_id == 1L) |>
  omopgenerics::newCohortTable(cohortSetRef = dplyr::tibble(
    cohort_definition_id = 1L, cohort_name = "flu"
  ))
cdm$target <- cdm$cohort2 |>
  omopgenerics::newCohortTable(cohortSetRef = dplyr::tibble(
    cohort_definition_id = c(1L, 2L, 3L),
    cohort_name = c("covid_test", "flu_test", "asthma")
  ))
```

Demographics

`addDemographics()` is used to characterise the demographics of a table. The table is needed to be part of a `cdm_reference` object and to contain a person identifier column (either person_id or subject_id). There are multiple columns that can be added with this function:

- *age*: the age at a certain `indexDate`. You can also add an *age group* column groupping individuals for different age group ranges.

DRAFT

- *sex*: the sex of the individual.
- *prior observation*: the number of days between start of observation and `indexDate`.
- *future observation*: the number of days between `indexDate` and end of observation.
- *date of birth*: the birth date of the individual.

An example to add the demographics to a mock cohort table is:

```
cdm$my_flu_cohort |>
  addDemographics(
    indexDate = "cohort_start_date",
    ageGroup = list("children" = c(0, 17), "adult" = c(18, Inf))
  ) |>
  dplyr::glimpse()
```

```
## Rows: ??
## Columns: 10
## Database: DuckDB v1.0.0 [root@Darwin 23.4.0:R 4.4.1/:memory:]
## $ cohort_definition_id <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1…
## $ subject_id           <int> 117, 818, 634, 729, 886, 245, 761, 385, 597, 53, …
## $ cohort_start_date    <date> 1959-06-03, 1936-09-03, 1979-08-13, 2010-03-
14, …
## $ cohort_end_date      <date> 1960-11-29, 1981-03-23, 2053-12-02, 2044-04-
28, …
## $ age                  <int> 27, 2, 16, 36, 58, 20, 38, 82, 94, 77, 18, 5, 5, …
## $ age_group            <chr> "adult", "children", "children", "adult", "adult"…
## $ sex                  <chr> "Female", "Female", "Female", "Male", "Male", "Ma…
## $ prior_observation    <int> 10015, 976, 6068, 13221, 21371, 7397, 13961, 3005…
## $ future_observation   <int> 768, 41462, 32133, 40592, 20967, 10212, 20892, 19…
## $ date_of_birth        <date> 1932-01-01, 1934-01-01, 1963-01-01, 1974-01-
01, …
```

For each one of the functionalities there exist individual functions: `addAge()`, `addSex()`, `addPriorObservation()`, `addFutureObservation()` and `addDateOfBirth()`.

Observation period id

The *observation_period* contains the period of time that an individual in the database is in observation. There might be multiple individual periods per person, but they can not overlap each other. When doing analysis it can be of interest knowing if a certain date is in observation, whether the individual will be in observation after a certain time, and from which observation period is an observation. To do so we have two functions:

- `addInObservation()` to identify if an individual is in observation in a certain *window* respect an *indexDate*.
- `addObservationPeriodId()` to identify in which observation period ordinal is that date from.

```
cdm$gibleed |>
  addInObservation(
    indexDate = "cohort_start_date",
    window = list("obs_index_date" = c(0, 0), "in_1_year" = c(365, 365)),
    nameStyle = "{window_name}"
  ) |>
  addObservationPeriodId() |>
  dplyr::glimpse()
```

```
## Rows: ??
## Columns: 7
```

DRAFT

```
## Database: DuckDB v1.0.0 [root@Darwin 23.4.0:R 4.4.1/:memory:]
## $ cohort_definition_id  <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, …
## $ subject_id            <int> 962, 1158, 4462, 351, 3556, 320, 1965, 2105, 259…
## $ cohort_start_date     <date> 1995-07-09, 2016-12-27, 1990-10-23, 2018-06-
28,…
## $ cohort_end_date       <date> 2019-06-14, 2017-02-15, 2018-04-27, 2018-06-
29,…
## $ obs_index_date        <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, …
## $ in_1_year             <int> 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, …
## $ observation_period_id <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, …
```

Query functions

Usually OMOP CDM instances are stored in SQL databases. The functions that we have seen take the original table add the new columns and save the result into a new table (`name` argument). Each function has its own homologous one (same name terminated with 'Query') that instead of saving the result to a table only returns a query to generate the table. For local instances both functions provide exactly the same result.

Intersections

`PatientProfiles` has 15 functions that are used to compute intersections between tables. Common functions parameters:

- `indexDate` Name of the column that contains the date that will be the origin time of our calculations.
- `censorDate` Name of the column that contains the date to censor the observation window.
- `window` Window of time respect to the index date that we will consider relevant events on.

There exist 4 different function types:

- *Flag*: It creates a new integer column that can have 3 possible values: `1` whether an event of interest is observed; `0` if the event is not observed; `NA` if the individual is not in observation within that window.
- *Count*: It creates a new integer column with the number of observed events, `NA` is reported if the individual is not in observation in that window.
- *Date*: It creates a new date column that contains the date of a certain event, `NA` is reported if the event is not observed or the individual is not in observation in that window.
- *Days*: It creates a new integer date with the time difference with a certain event, `NA` is reported if the event is not observed or the individual is not in observation in that window.

For the *Flag* and *Count* functions there are 2 extra parameters: - `targetStartDate` Nome of the column that identifies the start of the event. - `targetEndDate` Name of the end of the episode, if `NULL` the event is considered to start and end on the `targetStartDate`.

With the following code you can add the number of visits recorded in the prior year (`number_visits`) and a flag to see if there is a record of a asthma test any time prior to the index date.

```
cdm$cohort1 |>
  addTableIntersectCount(
    tableName = "visit_occurrence",
    window = c(-365, 0),
    nameStyle = "number_visits"
```

DRAFT

```
  ) |>
  addCohortIntersectFlag(
    cohortTableName = "cohort2",
    cohortId = 3,
    window = c(-Inf, 0),
    nameStyle = "prior_asthma"
  ) |>
  dplyr::glimpse()
```

```
## Rows: ??
## Columns: 6
## Database: DuckDB v1.0.0 [root@Darwin 23.4.0:R 4.4.1/:memory:]
## $ cohort_definition_id <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1…
## $ subject_id           <int> 117, 818, 634, 886, 245, 761, 597, 53, 124, 285, …
## $ cohort_start_date    <date> 1959-06-03, 1936-09-03, 1979-08-13, 1996-07-
06, …
## $ cohort_end_date      <date> 1960-11-29, 1981-03-23, 2053-12-02, 2048-10-
18, …
## $ number_visits        <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
## $ prior_asthma         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0…
```

For the *Date* and *Days* functions there are 2 extra parameters: - `targetDate` Name of the column that contains the event of interest. - `order` Whether we are interested with the "first" or "last" event in the window.

With the following code you would add which is the date of the next test (of flu or covid) after the index date:

```
cdm$cohort1 |>
  addCohortIntersectDate(
    targetCohortTable = "cohort2",
    targetCohortId = c(1, 2),
    window = c(1, Inf)
  ) |>
  dplyr::glimpse()
```

```
## Rows: ??
## Columns: 6
## Database: DuckDB v1.0.0 [root@Darwin 23.4.0:R 4.4.1/:memory:]
## $ cohort_definition_id <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1…
## $ subject_id           <int> 761, 597, 124, 285, 157, 348, 297, 919, 830, 741,…
## $ cohort_start_date    <date> 2037-03-23, 2093-08-24, 1981-12-13, 1993-01-
28, …
## $ cohort_end_date      <date> 2058-09-28, 2141-02-14, 2020-03-08, 2006-09-
29, …
## $ covid_test_1_to_inf  <date> 2077-08-29, NA, NA, NA, 2043-06-27, NA, NA, 2045…
## $ flu_test_1_to_inf    <date> NA, 2194-01-24, 2058-08-07, 2093-09-06, NA, 2041…
```

NOTE that each function has some arguments related to the intersecting target (cohort, concept or clinical table).

Summarise data

`summariseResult()` is a function that allow the user to summarise multiple columns into multiple estimates (see `availableEstimates()`) into a standard format output, see the below example:

```
cdm$cohort1 |>
```

DRAFT

```
  addCohortName() |>
  summariseResult(
    group = "cohort_name",
    strata = list("sex", c("sex", "prior_asthma")),
    variables = list(c("number_visits", "age"), c("covid_test_1_to_inf", "flu_test_1_to_
    estimates = list(c("median", "q25", "q75"), c("min", "max"))
  ) |>
  dplyr::glimpse()
```

```
## Rows: 84
## Columns: 13
## $ result_id        <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,…
## $ cdm_name         <chr> "PP_MOCK", "PP_MOCK", "PP_MOCK", "PP_MOCK", "PP_MOCK"…
## $ group_name       <chr> "cohort_name", "cohort_name", "cohort_name", "cohort_…
## $ group_level      <chr> "flu", "flu", "flu", "flu", "flu", "flu", "flu", "flu…
## $ strata_name      <chr> "overall", "overall", "overall", "overall", "overall"…
## $ strata_level     <chr> "overall", "overall", "overall", "overall", "overall"…
## $ variable_name    <chr> "number records", "number subjects", "age", "age", "a…
## $ variable_level   <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N…
## $ estimate_name    <chr> "count", "count", "median", "q25", "q75", "median", "…
## $ estimate_type    <chr> "integer", "integer", "integer", "integer", "integer"…
## $ estimate_value   <chr> "350", "350", "49", "23", "87", "0", "0", "0", "1945-
…
## $ additional_name  <chr> "overall", "overall", "overall", "overall", "overall"…
## $ additional_level <chr> "overall", "overall", "overall", "overall", "overall"…
```

Conclusions

The PatientProfiles R package provides functionality to assist researchers working with data mapped to the OMOP CDM format. By basing the package around this data model which has a known structure the package could be developed with simple interfaces yet deep functionality. The package has already been used in published studies (Català et al., 2024; Mercadé-Besora et al., 2024) and is freely available to be used in future research.

Funding information

Development of the PatientProfiles R package was funded by the European Medicines Agency as part of the Data Analysis and Real World Interrogation Network (DARWIN EU®). This manuscript represents the views of the DARWIN EU® Coordination Centre only and cannot be interpreted as reflecting the views of the European Medicines Agency or the European Medicines Regulatory Network.

References

Blacketer, C. (2025). *Definition and DDLs for the OMOP common data model (CDM)*.

Català, M., & Burn, E. (2024). *Omopgenerics: Methods and classes for the OMOP common data model*. https://darwin-eu.github.io/omopgenerics/, https://github.com/darwin-eu/omopgenerics

Català, M., Guo, Y., Du, M., Lopez-Guell, K., Burn, E., & Mercade-Besora, N. (2025). *PatientProfiles: Identify characteristics of patients in the OMOP common data model*. https://darwin-eu.github.io/PatientProfiles/, https://github.com/darwin-eu/PatientProfiles

DRAFT

Català, M., Mercadé-Besora, N., Kolde, R., Trinh, N. T. H., Roel, E., Burn, E., Rathod-Mistry, T., Kostka, K., Man, W. Y., Delmestri, A., Nordeng, H. M. E., Uusküla, A., Duarte-Salles, T., Prieto-Alhambra, D., & Jödicke, A. M. (2024). The effectiveness of COVID-19 vaccines to prevent long COVID symptoms: Staggered cohort study of data from the UK, spain, and estonia. *The Lancet Respiratory Medicine*, *12*, 226–240. https://doi.org/10.1016/S2213-2600(23)00414-9

Mercadé-Besora, N., Li, X., Kolde, R., Trinh, N. T., Sanchez-Santos, M. T., Man, W. Y., Roel, E., Reyes, C., Delmestri, A., Nordeng, H. M. E., Uusküla, A., Duarte-Salles, T., Prats, C., Prieto-Alhambra, D., Jödicke, A. M., & Català, M. (2024). The role of COVID-19 vaccines in preventing post-COVID-19 thromboembolic and cardiovascular complications. *Heart*, *110*(9), 635–643. https://doi.org/10.1136/heartjnl-2023-323483

Overhage, J. M., Ryan, P. B., Reich, C. G., Hartzema, A. G., & Stang, P. E. (2011). Validation of a common data model for active safety surveillance research. *Journal of the American Medical Informatics Association*, *19*(1), 54–60. https://doi.org/10.1136/amiajnl-2011-000376

Wickham, H. (2011). Testthat: Get started with testing. *The R Journal*, *3*, 5–10. https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J., Kuhn, M., Pedersen, T. L., Miller, E., Bache, S. M., Müller, K., Ooms, J., Robinson, D., Seidel, D. P., Spinu, V., … Yutani, H. (2019). Welcome to the tidyverse. *Journal of Open Source Software*, *4*(43), 1686. https://doi.org/10.21105/joss.01686

DRAFT