# Tidy R programming with the OMOP common data model

Edward Burn, Adam Black, Berta Raventós, Yuchen Guo, Mike Du, Kim López Güe

2024-03-06T00:00:00+00:00

# Table of contents

# Preface

## Is this book for me?

We've written this book for anyone interested in a working with databases mapped to the OMOP Common Data Model (CDM) in a tidyverse inspired approach. That is, human centered, consistent, composable, and inclusive (see https://design.tidyverse.org/unifying.html for more details on these principles).

New to the OMOP CDM? We'd recommend you pare this book with The Book of OHDSI

New to R? We recommend you compliment the book with R for data science

## Citation

TO ADD

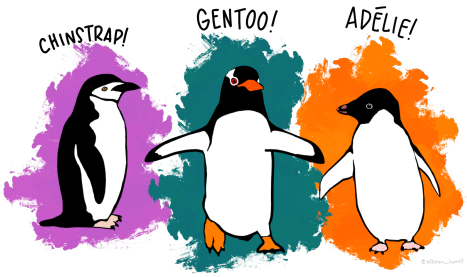
## License

## Code

The source code for the book can be found at this Github repository

# 1 Getting started

## 1.1 A first data analysis in R with a database




*Artwork by @allison_horst*

Before we start thinking about working with health care data spread across the OMOP common data model, let's first do a quick data analysis using a simpler dataset. For this we'll use data data from palmerpenguins package, which contains data on penguins collected from the Palmer Station in Antarctica.

## 1.2 Getting set up

Assuming that you have R and RStudio already set up, first we need to install a few packages not included in base R if we don´t already have them.

```
install.packages("dplyr")
install.packages("ggplot2")
install.packages("DBI")
install.packages("duckdb")
install.packages("palmerpenguins")
```

Once installed, we can load them like so.

```
library(dplyr)
library(ggplot2)
library(DBI)
library(duckdb)
library(palmerpenguins)
```

## 1.3 Taking a peek at the data

We can get an overview of the data using the `glimpse()` command.

```
glimpse(penguins)
```

```
Rows: 344
Columns: 8
$ species           <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Adel~
$ island            <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgerse~
$ bill_length_mm    <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, ~
$ bill_depth_mm     <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, ~
$ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186~
$ body_mass_g       <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, ~
$ sex               <fct> male, female, female, NA, female, male, female, male~
$ year              <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007~
```

Or we could take a look at the first rows of the data using `head()`

```
head(penguins, 5)
```

```
# A tibble: 5 x 8
  species island    bill_length_mm bill_depth_mm flipper_l~1 body_~2 sex    year
  <fct>   <fct>              <dbl>         <dbl>       <int>   <int> <fct> <int>
1 Adelie  Torgersen           39.1          18.7         181    3750 male   2007
2 Adelie  Torgersen           39.5          17.4         186    3800 fema~  2007
3 Adelie  Torgersen           40.3          18           195    3250 fema~  2007
4 Adelie  Torgersen           NA            NA            NA      NA <NA>   2007
5 Adelie  Torgersen           36.7          19.3         193    3450 fema~  2007
# ... with abbreviated variable names 1: flipper_length_mm, 2: body_mass_g
```

## 1.4 Inserting data into a database

Let's put our penguins data into a duckdb database. We create the duckdb database, add the penguins data, and then create a reference to the table containing the data.

```
db<-dbConnect(duckdb::duckdb(), dbdir=":memory:")
dbWriteTable(db, "penguins", penguins)
penguins_db<-tbl(db, "penguins")
```

Now the data is in a database we could use SQL to get the first rows that we saw before

```
dbGetQuery(db, "SELECT * FROM penguins LIMIT 5")
```

```
  species    island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g
1  Adelie Torgersen           39.1          18.7               181        3750
2  Adelie Torgersen           39.5          17.4               186        3800
3  Adelie Torgersen           40.3          18.0               195        3250
4  Adelie Torgersen             NA            NA                NA          NA
5  Adelie Torgersen           36.7          19.3               193        3450
     sex year
1   male 2007
2 female 2007
3 female 2007
4   <NA> 2007
5 female 2007
```

But we could also use the same R code as before

```
head(penguins_db, 5)
```

```
# Source:   SQL [5 x 8]
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/:memory:]
  species island    bill_length_mm bill_depth_mm flipper_l~1 body_~2 sex    year
  <fct>   <fct>              <dbl>         <dbl>       <int>   <int> <fct> <int>
1 Adelie  Torgersen           39.1          18.7         181    3750 male   2007
2 Adelie  Torgersen           39.5          17.4         186    3800 fema~  2007
3 Adelie  Torgersen           40.3          18           195    3250 fema~  2007
4 Adelie  Torgersen           NA            NA            NA      NA <NA>   2007
5 Adelie  Torgersen           36.7          19.3         193    3450 fema~  2007
# ... with abbreviated variable names 1: flipper_length_mm, 2: body_mass_g
```

## 1.5 Translation from R to SQL

The magic here is provided by dbplyr which takes the R code and converts it into SQL, which is this case looks like

```
head(penguins_db, 1) %>%
  show_query()
```

```
<SQL>
SELECT *
FROM penguins
LIMIT 1
```

More complicated SQL can also be written in what might be familiar dplyr code, for example

```
penguins_db %>%
  group_by(species) %>%
  summarise(min_bill_length_mm=min(bill_length_mm),
            median_bill_length_mm=median(bill_length_mm),
            max_bill_length_mm=max(bill_length_mm)) %>%
  mutate(min_max_bill_length_mm=paste0(min_bill_length_mm,
                                       " to ",
                                       max_bill_length_mm)) %>%
  select("species",
         "median_bill_length_mm",
         "min_max_bill_length_mm")
```

```
# Source:   SQL [3 x 3]
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/:memory:]
  species   median_bill_length_mm min_max_bill_length_mm
  <fct>                     <dbl> <chr>
1 Adelie                     38.8 32.1 to 46.0
2 Gentoo                     47.3 40.9 to 59.6
3 Chinstrap                  49.6 40.9 to 58.0
```

with the corresponding SQL looking like

```
penguins_db %>%
  group_by(species) %>%
  summarise(min_bill_length_mm=min(bill_length_mm),
```

```
            median_bill_length_mm=median(bill_length_mm),
            max_bill_length_mm=max(bill_length_mm)) %>%
  mutate(min_max_bill_length_mm=paste0(min, " to ", max)) %>%
  select("species",
         "median_bill_length_mm",
         "min_max_bill_length_mm") %>%
  show_query()
```

```
<SQL>
SELECT
  species,
  median_bill_length_mm,
  CONCAT_WS('', .Primitive("min"), ' to ', .Primitive("max")) AS min_max_bill_length_mm
FROM (
  SELECT
    species,
    MIN(bill_length_mm) AS min_bill_length_mm,
    PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY bill_length_mm) AS median_bill_length_mm,
    MAX(bill_length_mm) AS max_bill_length_mm
  FROM penguins
  GROUP BY species
) q01
```

## 1.6 Example analysis

Let´s start by getting a count by species

```
penguins_db %>%
  group_by(species) %>%
  count()
```

```
# Source:   SQL [3 x 2]
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/:memory:]
  species       n
  <fct>     <dbl>
1 Adelie      152
2 Gentoo      124
3 Chinstrap    68
```

Now suppose we are particularly interested in the body mass variable. We can first notice that there are a couple of missing records for this.

```
penguins_db %>%
  mutate(missing_body_mass_g=if_else(
    is.na(body_mass_g),1,0
  )) %>%
  group_by(species, missing_body_mass_g) %>%
  tally()
```

```
# Source:   SQL [5 x 3]
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/:memory:]
# Groups:   species
  species   missing_body_mass_g     n
  <fct>                   <dbl> <dbl>
1 Adelie                      0   151
2 Adelie                      1     1
3 Gentoo                      0   123
4 Gentoo                      1     1
5 Chinstrap                   0    68
```

We can get the mean for each of the species (dropping those two missing records).

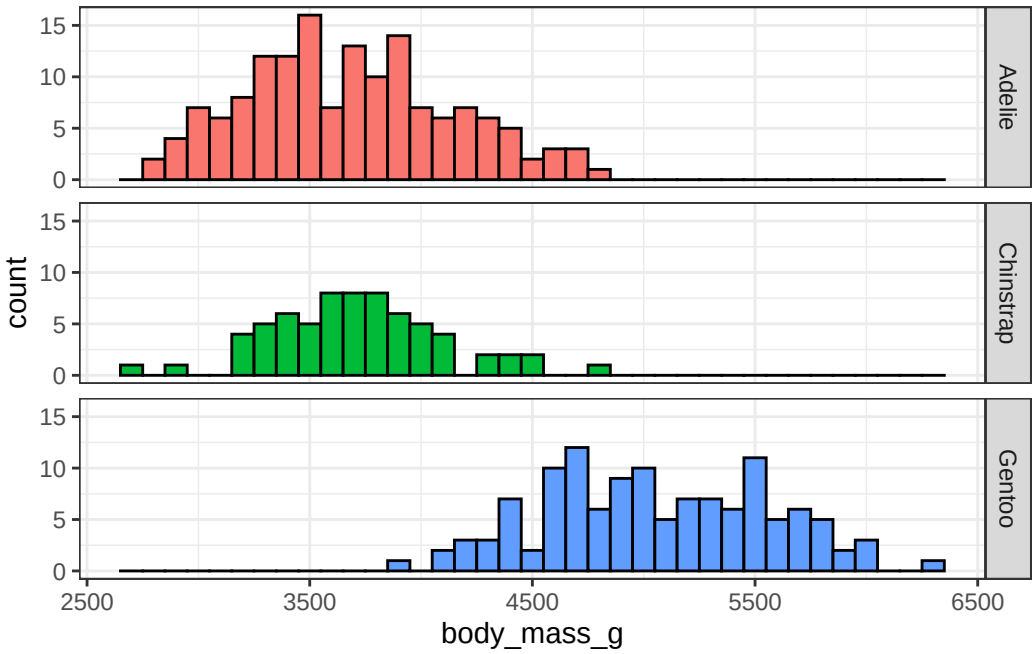
```
penguins_db %>%
  group_by(species) %>%
  summarise(mean_body_mass_g=round(mean(body_mass_g, na.rm=TRUE),0))
```

```
# Source:   SQL [3 x 2]
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/:memory:]
  species   mean_body_mass_g
  <fct>                <dbl>
1 Adelie                3701
2 Gentoo                5076
3 Chinstrap             3733
```

We can then also do a histogram for each of the species. For this we need to bring the data into R so that we can work with `ggplot()`, and we use `collect()` to do this.

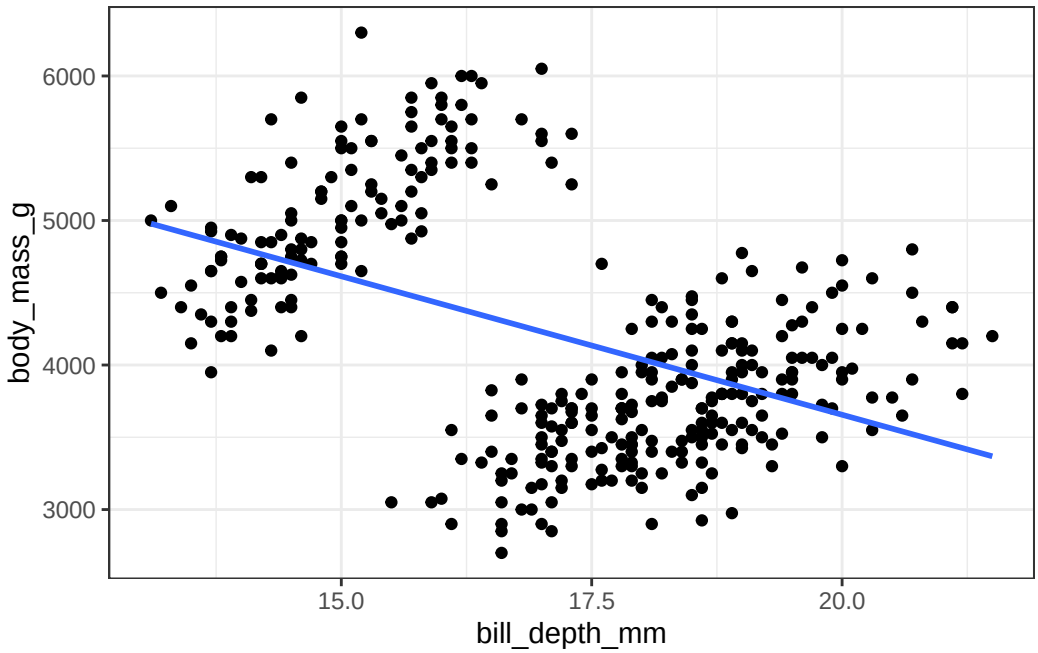
```
penguins_db %>%
  collect() %>%
```

```
  ggplot(aes(group=species, fill=species))+
  facet_grid(species~ .) +
  geom_histogram(aes(body_mass_g), colour="black", binwidth = 100)+
  theme_bw()+
  theme(legend.position = "none")
```



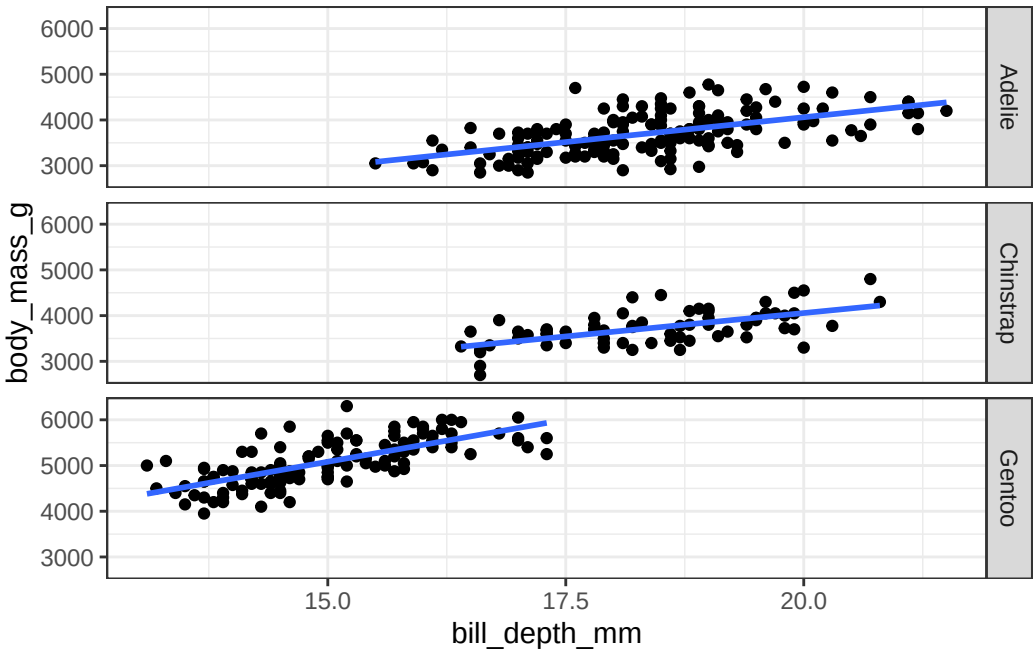

How about the relationship between body mass and bill depth?

```
penguins %>%
  collect() %>%
  ggplot(aes(x=bill_depth_mm,y=body_mass_g))+
  geom_point()+
  geom_smooth(method="lm",se=FALSE )+
  theme_bw()+
  theme(legend.position = "none")
```




But what about by species?

```
penguins %>%
  collect() %>%
  ggplot(aes(x=bill_depth_mm,y=body_mass_g))+
  facet_grid(species~ .) +
  geom_point()+
  geom_smooth(method="lm",se=FALSE )+
  theme_bw()+
  theme(legend.position = "none")
```




As well as having an example of working with data in database from R, you also have an example of Simpson´s paradox! And now we've reached the end of this example, we can close the database like so

## 1.7 Further reading

- R for Data Science (Chapter 13: Relational data)
- Writing SQL with dbplyr
- Data Carpentry: SQL databases and R

# 2 Creating a reference to a database using the OMOP common data model

## 2.1 Connecting to a database from R using DBI

Database connections from R can be made using the DBI package. The back-end for `DBI` is facilitated by database specific driver packages. As an example, lets say we want to work with a local duckdb from R. In this case the we can use the duckdb R package as the driver.

```
library(DBI)
db<-dbConnect(duckdb::duckdb(), dbdir=":memory:")
```

If we instead wanted to connect to other database management systems, these connections would be supported by the associated back-end packages and could look something like the below example for Postgres:

```
# Postgres
db <- DBI::dbConnect(RPostgres::Postgres(),
                     dbname = Sys.getenv("CDM5_POSTGRESQL_DBNAME"),
                     host = Sys.getenv("CDM5_POSTGRESQL_HOST"),
                     user = Sys.getenv("CDM5_POSTGRESQL_USER"),
                     password = Sys.getenv("CDM5_POSTGRESQL_PASSWORD"))
```

## 2.2 Creating a reference to the OMOP common data model

As seen in the previous chapter, once a connection to the database has been created then we could create references to the various tables in the database and build queries using in a familiar dplyr style. However, as we already know what the structure of the OMOP CDM looks like, we can avoid the overhead of building *ad hoc* references by instead using the `CDMConnector` package to quickly create a reference to the OMOP CDM data as a whole.

If you don't already have it installed, the first step would be to install `CDMConnector` from CRAN.

```
install.packages("CDMConnector")
```

Once we have it installed, we can then load it as with other R packages.

```
library(CDMConnector)
```

For this example, we'll use the Eunomia example data contained in a duckdb database. First we need to download the data. And once downloaded, make sure to add the path to your Renviron.

```
# change pathToData to the location you want to save the data
CDMConnector::downloadEunomiaData(
  pathToData = here::here(),
  overwrite = TRUE
)
# once downloaded, save your pathToData to your Renviron (and then restart R)
# EUNOMIA_DATA_FOLDER="......"
```

```
db <- DBI::dbConnect(duckdb::duckdb(),
                     dbdir = CDMConnector::eunomia_dir())
cdm <- CDMConnector::cdm_from_con(con = db,
                                  cdm_schema = "main")
cdm
```

```
# OMOP CDM reference (tbl_duckdb_connection)

Tables: person, observation_period, visit_occurrence, visit_detail, condition_occurrence, dru
```

Once we have created the our reference to the overall OMOP CDM, we can reference specific tables using the "$" operator or [[" "]].

```
cdm$observation_period
```

```
# Source:   table<main.observation_period> [?? x 5]
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtmp
   observation_period_id person_id observation_period_start~1 observat~2 perio~3
                   <dbl>     <dbl> <date>                     <date>       <dbl>
 1                     6         6 1963-12-31                 2007-02-06  4.48e7
 2                    13        13 2009-04-26                 2019-04-14  4.48e7
 3                    27        27 2002-01-30                 2018-11-21  4.48e7
```

```
 4                   16        16 1971-10-14                 2017-11-02  4.48e7
 5                   55        55 2009-05-30                 2019-03-23  4.48e7
 6                   60        60 1990-11-21                 2019-01-23  4.48e7
 7                   42        42 1909-11-03                 2019-03-13  4.48e7
 8                   33        33 1986-05-12                 2018-09-10  4.48e7
 9                   18        18 1965-11-17                 2018-11-07  4.48e7
10                   25        25 2007-03-18                 2019-04-07  4.48e7
# ... with more rows, and abbreviated variable names
#   1: observation_period_start_date, 2: observation_period_end_date,
#   3: period_type_concept_id
```

```
cdm[["observation_period"]]
```

```
# Source:   table<main.observation_period> [?? x 5]
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtmp
   observation_period_id person_id observation_period_start~1 observat~2 perio~3
                   <dbl>     <dbl> <date>                     <date>       <dbl>
 1                     6         6 1963-12-31                 2007-02-06  4.48e7
 2                    13        13 2009-04-26                 2019-04-14  4.48e7
 3                    27        27 2002-01-30                 2018-11-21  4.48e7
 4                    16        16 1971-10-14                 2017-11-02  4.48e7
 5                    55        55 2009-05-30                 2019-03-23  4.48e7
 6                    60        60 1990-11-21                 2019-01-23  4.48e7
 7                    42        42 1909-11-03                 2019-03-13  4.48e7
 8                    33        33 1986-05-12                 2018-09-10  4.48e7
 9                    18        18 1965-11-17                 2018-11-07  4.48e7
10                    25        25 2007-03-18                 2019-04-07  4.48e7
# ... with more rows, and abbreviated variable names
#   1: observation_period_start_date, 2: observation_period_end_date,
#   3: period_type_concept_id
```

When we created our reference we could have also specified a subset of cdm tables that we want to read:

```
cdm <- CDMConnector::cdm_from_con(db,
                                  cdm_tables = c("person","observation_period"))
cdm
```

```
# OMOP CDM reference (tbl_duckdb_connection)

Tables: person, observation_period
```

Moreover, we can also specify a write schema and the tables that we are interested in it when creating our reference. For example, if we wanted to create a reference to the person and observation period tables in the common data model along with cohort tables in a schema we have write access to, we could do this like so:

```
cdm <- CDMConnector::cdm_from_con(db,
  cdm_schema = "main",
  cdm_tables = c("person","observation_period"),
  write_schema = "results",
  cohort_tables = c("exposure_cohort", "outcome_cohort"))
```

## 2.3 Database snapshot

We can also use `CDMConnector` to provide a summary of the metadata for the OMOP CDM data we have connected to

```
cdm_from_con(con = db,
             cdm_schema = "main") %>%
  snapshot() %>%
  glimpse()
```

```
List of 7
 $ cdm_source_name       : chr "Synthea synthetic health database"
 $ cdm_version           : chr "v5.3.1"
 $ cdm_holder            : chr "OHDSI Community"
 $ cdm_release_date      : Date[1:1], format: "2019-05-25"
 $ vocabulary_version    : chr "v5.0 18-JAN-19"
 $ person_cnt            : num 2694
 $ observation_period_cnt: num 5343
 - attr(*, "class")= chr "cdm_snapshot"
```

## 2.4 Further reading

- CDMConnector package

# 3 Exploring the CDM

Let's first connect again to our Eunomia data and create the reference to the common data model.

```
library(dbplyr)
library(dplyr)
library(CDMConnector)
library(ggplot2)
```

## 3.1 Counting people

The OMOP CDM is person-centric, with the person table containing records to uniquely identify each person in the database. As each row refers to a unique person, we can quickly get a count of the number of individuals in the database like so

```
cdm$person %>%
  count() %>%
  pull()
```

```
[1] 2694
```

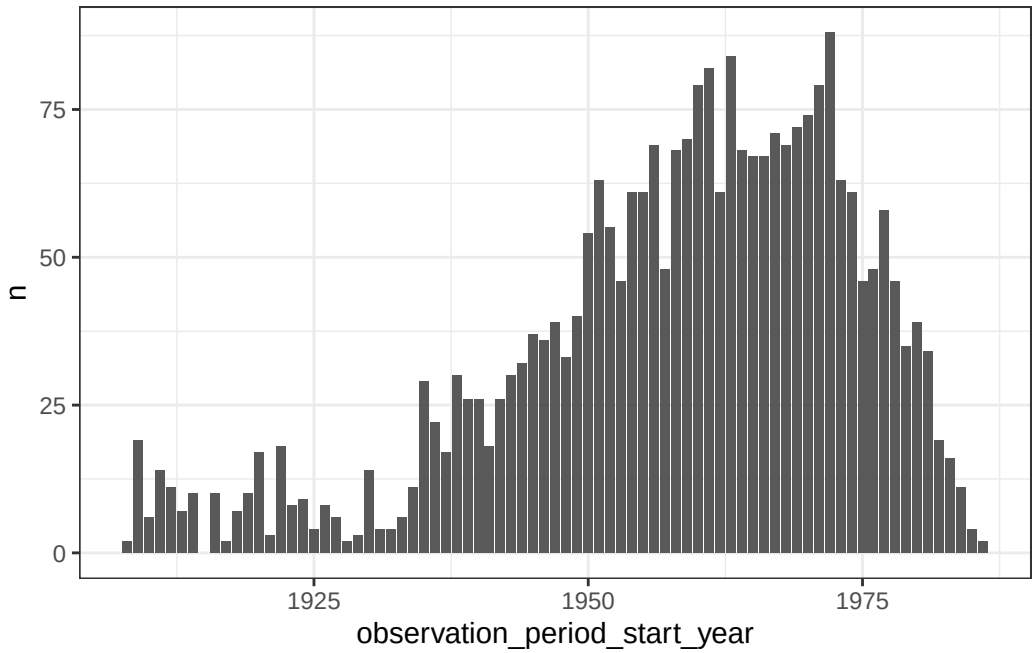
The person table also contains some demographic information, including a gender_concept_id for each person. We can get a count grouped by this variable, but as this uses a concept we'll also need to join to the concept table to get the corresponding concept name for each concept id.

```
cdm$person %>%
  group_by(gender_concept_id) %>%
  count() %>%
  left_join(cdm$concept,
            by=c("gender_concept_id" = "concept_id")) %>%
              select("gender_concept_id", "concept_name", "n") %>%
  collect()
```

```
# A tibble: 2 x 3
  gender_concept_id concept_name     n
              <dbl> <chr>        <dbl>
1              8532 FEMALE        1373
2              8507 MALE          1321
```

The observation period table contains records indicating spans of time over which clinical events can be reliably observed for the people in the person table. Someone can potentially have multiple observation periods. So say we wanted a count of people grouped by the year during which their first observation period started. We could do this as below (note the use of compute() to store the results of the first query in a temporary table in the database)

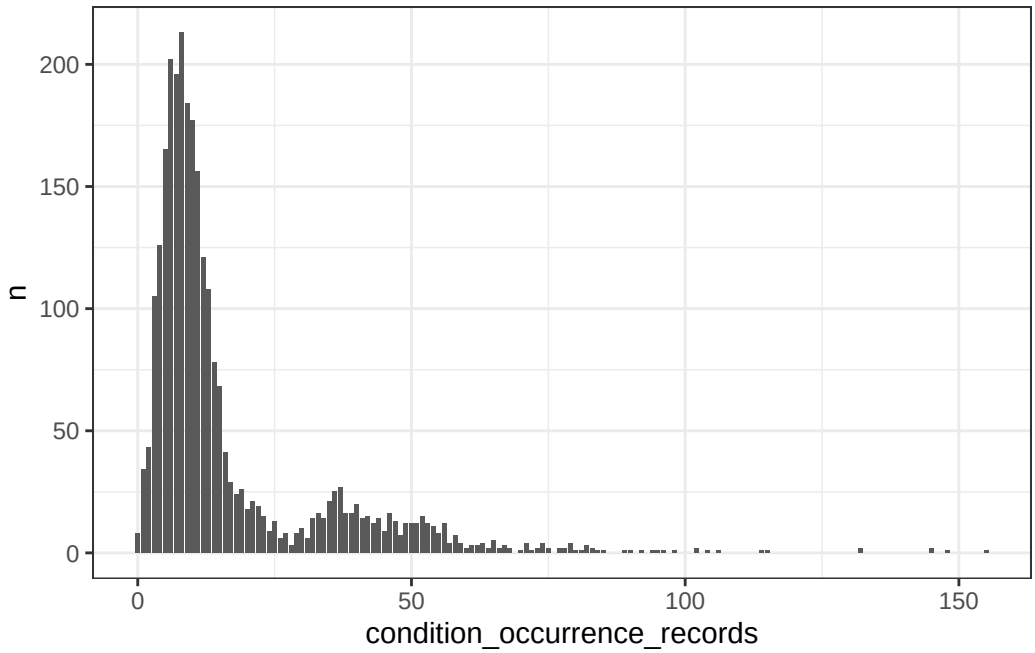
```
first_observation_period <- cdm$observation_period %>%
    group_by(person_id) %>%
    mutate(seq = dplyr::row_number()) %>%
    filter(seq==1) %>%
    compute()

cdm$person %>%
  left_join(first_observation_period,
            by = "person_id") %>%
  mutate(observation_period_start_year=year(observation_period_start_date)) %>%
  group_by(observation_period_start_year) %>%
  count() %>%
  collect() %>%
  ggplot() +
  geom_col(aes(observation_period_start_year, n)) +
  theme_bw()
```




## 3.2 Counting records

Number of drug exposure records per person

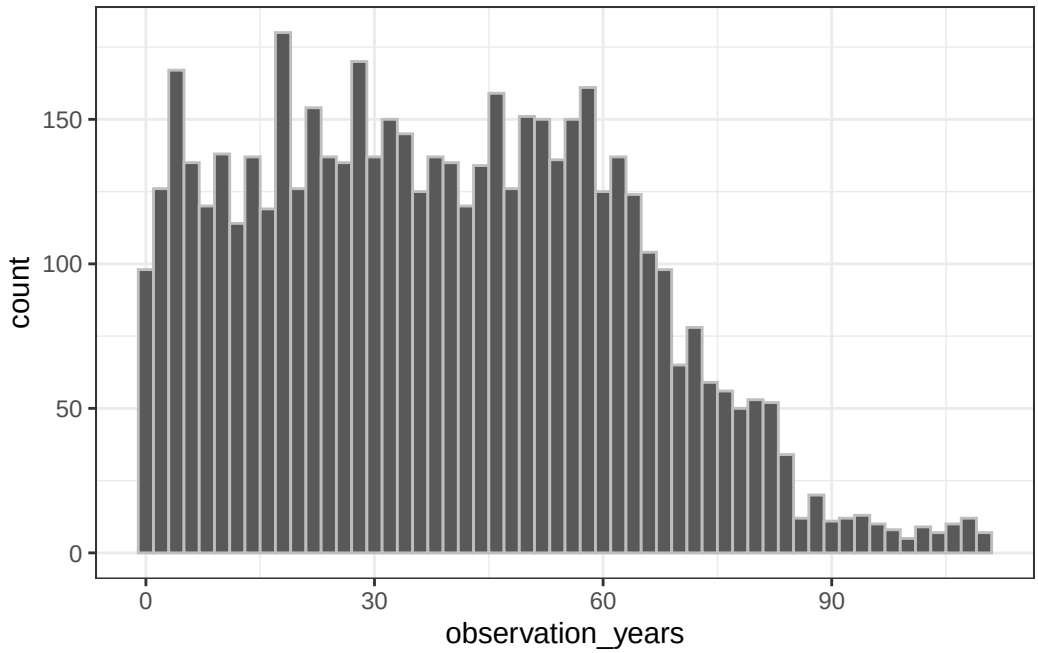
```
cdm$person %>%
  left_join(cdm$measurement %>%
  group_by(person_id) %>%
  count(name = "condition_occurrence_records"),
            by="person_id") %>%
  mutate(condition_occurrence_records = if_else(
    is.na(condition_occurrence_records), 0,
    condition_occurrence_records)) %>%
  group_by(condition_occurrence_records) %>%
  count() %>%
  collect() %>%
  ggplot() +
  geom_col(aes(condition_occurrence_records, n)) +
  theme_bw()
```




## 3.3 Working with dates

Dates are supported somewhat inconsistently by dbplyr, but CDMConnector provides some functions that provide more general support. We can use the datediff function from CDMConnector for example to calculate the difference between two dates. We can use this, for example, to get the number of years people's observation period last for.

```
cdm$observation_period %>%
  dplyr::mutate(observation_years =
                  !!CDMConnector::datediff("observation_period_start_date",
                                           "observation_period_end_date",
                                           interval = "year"))  %>%
  collect() %>%
  ggplot() +
  geom_histogram(aes(observation_years),
                 binwidth=2, colour="grey") +
  theme_bw()
```




## 3.4 Statistical summaries

We can also use summarise for various other calculations

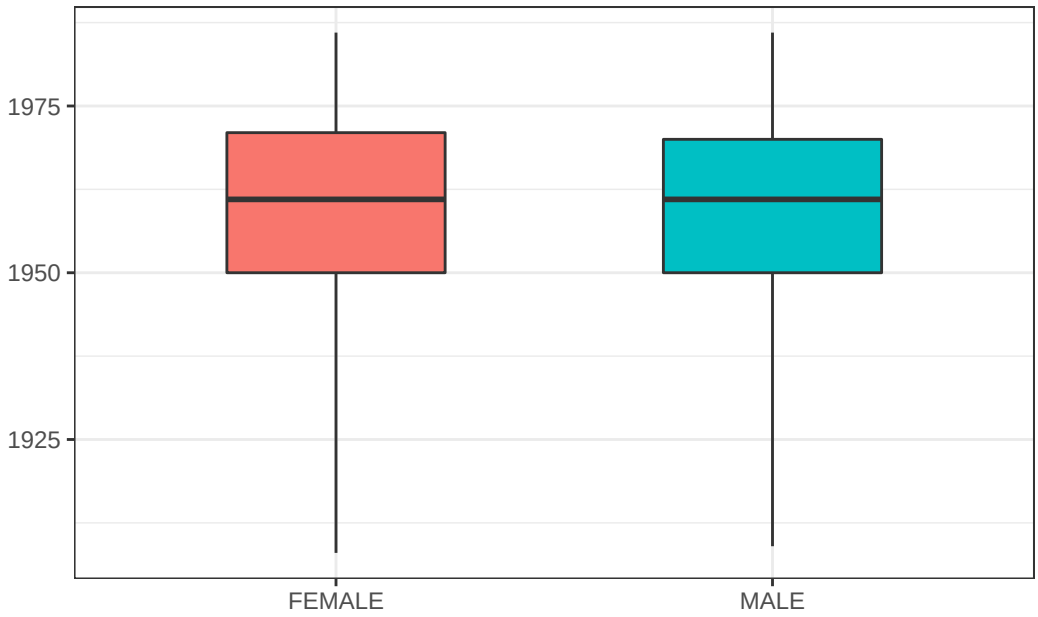
```
cdm$person %>%
  summarise(min_year_of_birth = min(year_of_birth, na.rm=TRUE),
            q05_year_of_birth = quantile(year_of_birth, 0.05, na.rm=TRUE),
            mean_year_of_birth = round(mean(year_of_birth, na.rm=TRUE),0),
            median_year_of_birth = median(year_of_birth, na.rm=TRUE),
            q95_year_of_birth = quantile(year_of_birth, 0.95, na.rm=TRUE),
            max_year_of_birth = max(year_of_birth, na.rm=TRUE)) %>%
  glimpse()
```

```
Rows: ??
Columns: 6
Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\RtmpA
$ min_year_of_birth    <dbl> 1908
$ q05_year_of_birth    <dbl> 1922
$ mean_year_of_birth   <dbl> 1958
$ median_year_of_birth <dbl> 1961
$ q95_year_of_birth    <dbl> 1979
```

```
$ max_year_of_birth    <dbl> 1986
```

As we've seen before, we can also quickly get results for various groupings or restrictions

```
cdm$person %>%
   group_by(gender_concept_id) %>%
   summarise(min_year_of_birth = min(year_of_birth, na.rm=TRUE),
             q25_year_of_birth = quantile(year_of_birth, 0.25, na.rm=TRUE),
             median_year_of_birth = median(year_of_birth, na.rm=TRUE),
             q75_year_of_birth = quantile(year_of_birth, 0.75, na.rm=TRUE),
             max_year_of_birth = max(year_of_birth, na.rm=TRUE)) %>%
  left_join(cdm$concept,
             by=c("gender_concept_id" = "concept_id")) %>%
   collect() %>%
  ggplot(aes(x = concept_name, group = concept_name,
              fill = concept_name)) +
  geom_boxplot(aes(
    lower = q25_year_of_birth,
    upper = q75_year_of_birth,
    middle = median_year_of_birth,
    ymin = min_year_of_birth,
    ymax = max_year_of_birth),
    stat = "identity", width = 0.5) +
  theme_bw()+
  theme(legend.position = "none") +
  xlab("")
```


1975
1950
1925
FEMALE
MALE

# 4 Indentifying patient characteristics

For this chapter, we'll again use our example COVID-19 dataset.

```
library(DBI)
library(dbplyr)
library(dplyr)
library(here)
library(CDMConnector)
library(PatientProfiles)
library(ggplot2)
```

```
db<-dbConnect(duckdb::duckdb(),
              dbdir = eunomiaDir(datasetName = "synthea-covid19-10k"))
cdm <- cdm_from_con(db,
                    cdm_schema = "main")
```

As part of an analysis we almost always have a need to identify certain characteristics related to the individuals in our data. These characteristics might be time-invariant (ie a characteristic that does not change as time passes and a person ages) or time-varying. In various datasets, however, characteristics that could conceptually be considered as time-varying are encoded as time-invariant. One example for the latter is that in some cases an individual may be associated with a particular socioeconomic status or nationality that for the purposes of the data is treated as time-invariant.

## 4.1 Adding specific demographics

The `PatientProfiles` package makes it easy for us to add demographic information to tables in the OMOP CDM. Say we are interested in individuals age and sex at time of diagnosis with COVID-19, we can add these variables to the table like so. Note that because age is time-varying, we have to specify the variable with the date for which we want to calculate age relative to.

```
cdm$condition_occurrence <- cdm$condition_occurrence %>%
  addSex() %>%
  addAge(indexDate = "condition_start_date")

cdm$condition_occurrence %>%
  glimpse()
```

```
Rows: ??
Columns: 18
Database: DuckDB 0.8.1 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtmpo
$ condition_occurrence_id       <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1~
$ person_id                     <int> 2, 6, 7, 8, 8, 8, 8, 16, 16, 18, 18, 25,~
$ condition_concept_id          <int> 381316, 321042, 381316, 37311061, 437663~
$ condition_start_date          <date> 1986-09-08, 2021-06-23, 2021-04-07, 202~
$ condition_start_datetime      <dttm> 1986-09-08, 2021-06-23, 2021-04-07, 202~
$ condition_end_date            <date> 1986-09-08, 2021-06-23, 2021-04-07, 202~
$ condition_end_datetime        <dttm> 1986-09-08, 2021-06-23, 2021-04-07, 202~
$ condition_type_concept_id     <int> 38000175, 38000175, 38000175, 38000175, ~
$ condition_status_concept_id   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
$ stop_reason                   <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
$ provider_id                   <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
$ visit_occurrence_id           <int> 19, 55, 67, 79, 79, 79, 79, 168, 171, 19~
$ visit_detail_id               <int> 1000019, 1000055, 1000067, 1000079, 1000~
$ condition_source_value        <chr> "230690007", "410429000", "230690007", "~
$ condition_source_concept_id   <int> 381316, 321042, 381316, 37311061, 437663~
$ condition_status_source_value <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
$ sex                           <chr> "Female", "Male", "Male", "Male", "Male"~
$ age                           <dbl> 57, 25, 97, 2, 2, 2, 2, 75, 77, 57, 76, ~
```

We now have two variables added containing values for age and sex.

```
cdm$condition_occurrence %>%
  glimpse()
```

```
Rows: ??
Columns: 18
Database: DuckDB 0.8.1 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtmpo
$ condition_occurrence_id       <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1~
$ person_id                     <int> 2, 6, 7, 8, 8, 8, 8, 16, 16, 18, 18, 25,~
$ condition_concept_id          <int> 381316, 321042, 381316, 37311061, 437663~
```

```
$ condition_start_date          <date> 1986-09-08, 2021-06-23, 2021-04-07, 202~
$ condition_start_datetime      <dttm> 1986-09-08, 2021-06-23, 2021-04-07, 202~
$ condition_end_date            <date> 1986-09-08, 2021-06-23, 2021-04-07, 202~
$ condition_end_datetime        <dttm> 1986-09-08, 2021-06-23, 2021-04-07, 202~
$ condition_type_concept_id     <int> 38000175, 38000175, 38000175, 38000175, ~
$ condition_status_concept_id   <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
$ stop_reason                   <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
$ provider_id                   <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
$ visit_occurrence_id           <int> 19, 55, 67, 79, 79, 79, 79, 168, 171, 19~
$ visit_detail_id               <int> 1000019, 1000055, 1000067, 1000079, 1000~
$ condition_source_value        <chr> "230690007", "410429000", "230690007", "~
$ condition_source_concept_id   <int> 381316, 321042, 381316, 37311061, 437663~
$ condition_status_source_value <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
$ sex                           <chr> "Female", "Male", "Male", "Male", "Male"~
$ age                           <dbl> 57, 25, 97, 2, 2, 2, 2, 75, 77, 57, 76, ~
```

And with these now added it is straightforward to calculate mean age at condition start date by sex.

```
cdm$condition_occurrence %>%
  summarise(mean_age = mean(age, na.rm=TRUE), .by = "sex")
```

```
# Source:   SQL [2 x 2]
# Database: DuckDB 0.8.1 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtmp
  sex    mean_age
  <chr>     <dbl>
1 Female     50.8
2 Male       56.5
```

## 4.2 Adding multiple demographics simultaneously

We've now seen individual functions from `PatientProfiles` to add age and sex, and the package has others to add other characteristics like days of prior history in the database (`PatientProfiles::addPriorObservation()`). In additional to these individuals functions, the package also provides a more general function to get all of these characteristics at the same time (that is more time efficient that getting them one by one).

```
cdm$drug_exposure <- cdm$drug_exposure %>%
  addDemographics(indexDate = "drug_exposure_start_date")
```

```
cdm$drug_exposure %>%
  glimpse()
```

```
Rows: ??
Columns: 27
Database: DuckDB 0.8.1 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtmpo
$ drug_exposure_id             <int> 245761, 245762, 245763, 245764, 245765, 2~
$ person_id                    <int> 7764, 7764, 7764, 7764, 7764, 7764, 7764,~
$ drug_concept_id              <int> 40213227, 40213201, 40213198, 40213154, 4~
$ drug_exposure_start_date     <date> 2015-02-08, 2010-01-10, 2010-01-10, 2017~
$ drug_exposure_start_datetime <dttm> 2015-02-08 22:40:04, 2010-01-10 22:40:04~
$ drug_exposure_end_date       <date> 2015-02-08, 2010-01-10, 2010-01-10, 2017~
$ drug_exposure_end_datetime   <dttm> 2015-02-08 22:40:04, 2010-01-10 22:40:04~
$ verbatim_end_date            <date> 2015-02-08, 2010-01-10, 2010-01-10, 2017~
$ drug_type_concept_id         <int> 32869, 32869, 32869, 32869, 32869, 32869,~
$ stop_reason                  <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ refills                      <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
$ quantity                     <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
$ days_supply                  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
$ sig                          <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ route_concept_id             <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,~
$ lot_number                   <chr> "0", "0", "0", "0", "0", "0", "0", "0", "~
$ provider_id                  <int> 14656, 14656, 14656, 14656, 14656, 14656,~
$ visit_occurrence_id          <int> 80896, 80891, 80891, 80892, 80895, 80896,~
$ visit_detail_id              <int> 1080896, 1080891, 1080891, 1080892, 10808~
$ drug_source_value            <chr> "113", "33", "133", "140", "140", "140", ~
$ drug_source_concept_id       <int> 40213227, 40213201, 40213198, 40213154, 4~
$ route_source_value           <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ dose_unit_source_value       <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, N~
$ age                          <dbl> 71, 66, 66, 73, 72, 71, 69, 67, 70, 68, 6~
$ sex                          <chr> "Male", "Male", "Male", "Male", "Male", "~
$ prior_observation            <dbl> 2597, 742, 742, 3339, 2968, 2597, 1855, 1~
$ future_observation           <dbl> 896, 2751, 2751, 154, 525, 896, 1638, 238~
```

With these characteristics now all added, we can now calculate mean age, prior observation (how many days have passed since the individual's most recent observation start date), and future observation (how many days until the individual's nearest observation end date) at drug exposure start date by sex.

```
cdm$drug_exposure %>%
  summarise(mean_age = mean(age, na.rm=TRUE),
```

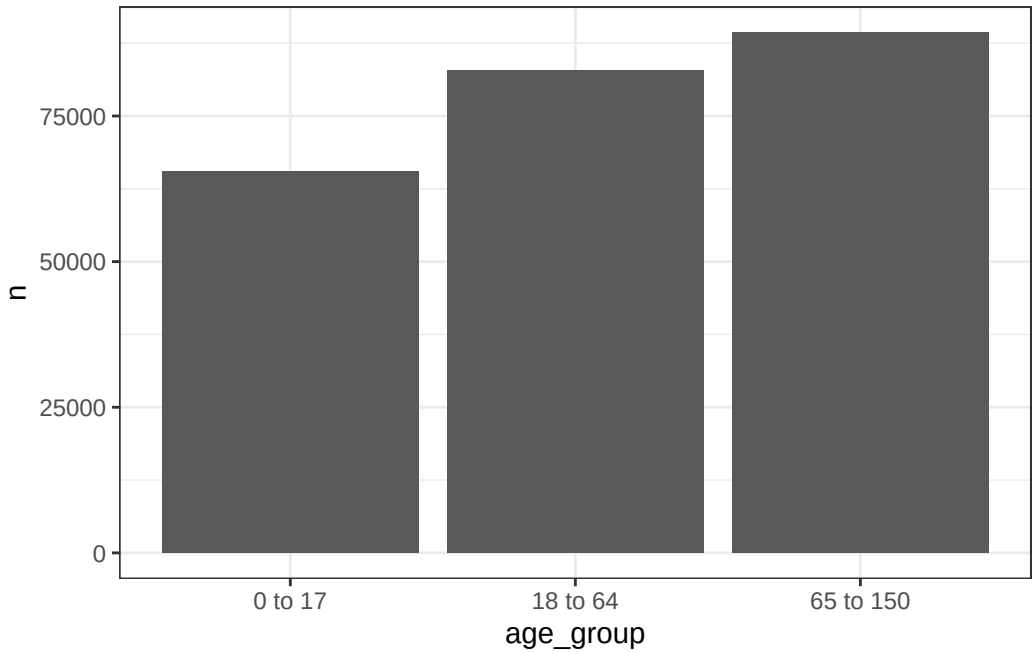
```
            mean_prior_observation = mean(prior_observation, na.rm=TRUE),
            mean_future_observation = mean(future_observation, na.rm=TRUE),
            .by = "sex")
```

```
# Source:   SQL [2 x 4]
# Database: DuckDB 0.8.1 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtmp
  sex    mean_age mean_prior_observation mean_future_observation
  <chr>     <dbl>                  <dbl>                   <dbl>
1 Male       43.0                  2455.                   1768.
2 Female     39.4                  2096.                   1661.
```

## 4.3 Creating categories

When we add age, either via addAge or addDemographics, we can also add another variable containing age groups. These age groups are specified in a list of vectors, each of which contain the lower and upper bounds.

```
cdm$visit_occurrence <- cdm$visit_occurrence %>%
  addAge(indexDate = "visit_start_date",
    ageGroup = list(c(0,17), c(18, 64),
                    c(65, 150)))

cdm$visit_occurrence %>%
  filter(age >= 0 & age <= 150) %>%
  group_by(age_group) %>%
  tally() %>%
  collect() %>%
  ggplot() +
  geom_col(aes(x = age_group, y = n)) +
  theme_bw()
```




`PatientProfiles` also provides a more general function for adding categories. Can you guess it's name? That's right, we have `PatientProfiles::addCategories()` for this.

```
cdm$condition_occurrence %>%
  addPriorObservation(indexDate = "condition_start_date") %>%
  addCategories(
    variable = "prior_observation",
    categories = list("prior_observation_group" = list(
      c(0, 364), c(365, 999999) # Inf not currently supported as an upper bound
    ))
  )
```

```
# Source:   table<dbplyr_007> [?? x 20]
# Database: DuckDB 0.8.1 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtmp
   condition_occurrence_id person_id condition_concept_id condition_start_date
                     <int>     <int>                <int> <date>
 1                       1         2               381316 1986-09-08
 2                       2         6               321042 2021-06-23
 3                       3         7               381316 2021-04-07
 4                       4         8             37311061 2021-01-08
 5                       5         8               437663 2021-01-08
 6                       6         8              4089228 2021-01-08
```

```
 7                      7         8              254761 2021-01-08
 8                      8        16              381316 2020-02-11
 9                      9        16              313217 2021-10-05
10                     10        18              317576 1993-08-08
# i more rows
# i 16 more variables: condition_start_datetime <dttm>,
#   condition_end_date <date>, condition_end_datetime <dttm>,
#   condition_type_concept_id <int>, condition_status_concept_id <int>,
#   stop_reason <chr>, provider_id <int>, visit_occurrence_id <int>,
#   visit_detail_id <int>, condition_source_value <chr>,
#   condition_source_concept_id <int>, condition_status_source_value <chr>, ...
```

## 4.4 Adding custom variables

While `PatientProfiles` provides a range of functions that can help add characteristics of interest, you may want to add other, custom features. Obviously we can't cover here all possible custom characteristics you may wish to add. However, custom features do generally come in two forms.

The first is where we want to add a new variable derived from other variables in our table. Here we'll be using `dplyr::mutate()`. For example, perhaps we just want to add a new variable to our observation period table containing the year of individuals' observation period start date.

```
cdm$observation_period <- cdm$observation_period %>%
  mutate(observation_period_start_year = year(observation_period_start_date))

cdm$observation_period %>%
  glimpse()
```

```
Rows: ??
Columns: 6
Database: DuckDB 0.8.1 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtmpo
$ observation_period_id         <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1~
$ person_id                     <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 1~
$ observation_period_start_date <date> 2014-05-09, 1977-04-11, 2014-04-19, 201~
$ observation_period_end_date   <date> 2023-05-12, 1986-09-15, 2023-04-22, 202~
$ period_type_concept_id        <int> 44814724, 44814724, 44814724, 44814724, ~
$ observation_period_start_year <dbl> 2014, 1977, 2014, 2014, 2013, 2013, 2013~
```

The second, normally more complex task, is adding a new variable that involves joining to some other table. This table may well have been created by some intermediate query that we wrote to derive the variable of interest. For example, lets say we want to add each number of condition occurrence records for each individual to the person table (remember that we saw how to calculate this in the previous chapter). Here we'll also create this as a new table containing just the information we're interested in and compute to a temporary table.

```
condition_summary <- cdm$person %>%
  left_join(cdm$condition_occurrence %>%
  group_by(person_id) %>%
  count(name = "condition_occurrence_records"),
  by="person_id") %>%
  select("person_id", "condition_occurrence_records") %>%
  mutate(condition_occurrence_records = if_else(
    is.na(condition_occurrence_records),
    0, condition_occurrence_records)) %>%
  computeQuery()

condition_summary %>%
  glimpse()
```

```
Rows: ??
Columns: 2
Database: DuckDB 0.8.1 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtmpo
$ person_id                   <int> 2, 6, 7, 8, 16, 18, 25, 36, 40, 44, 47, 5~
$ condition_occurrence_records <dbl> 1, 1, 1, 4, 2, 2, 1, 4, 1, 2, 5, 1, 3, 2,~
```

Taking care with joins

When adding variables through joins we need to pay particular attention to the dimensions of the resulting table. While sometimes we may want to have additional rows added as well as new columns, this is often not desired. If we, for example, have a table with one row per person then a left join to a table with multiple rows per person will result in a table with f we, for example, have a table with one row per person then a left join to a table with multiple rows per person (unless those people with more than record are only in the second table).
Examples where to be careful include when joining to the observation period table, as individuals can have multiple observation periods, and when working with cohorts (Which are the focus of the next chapter), as individuals can also enter the same study cohort multiple times.
Just to underline how problematic joins can become if we don't take care, here we join

the condition occurrence table and the drug exposure table both of which have multiple records per person. Remember this is just with our small synthetic data, so when working with real patient data which is oftentimes much, much larger this would be extremely problematic (and would unlikely be needed to answer any research question). In other words, don't try this at home!

```
cdm$condition_occurrence %>%
  tally()
```

```
# Source:   SQL [1 x 1]
# Database: DuckDB 0.8.1 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rt
      n
  <dbl>
1  9967
```

```
cdm$drug_exposure %>%
  tally()
```

```
# Source:   SQL [1 x 1]
# Database: DuckDB 0.8.1 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rt
       n
   <dbl>
1 337509
```

```
cdm$condition_occurrence %>%
  select(person_id, condition_start_date) %>%
  left_join(cdm$drug_exposure %>%
  select(person_id, drug_exposure_start_date),
  by = "person_id") %>%
  tally()
```

```
# Source:   SQL [1 x 1]
# Database: DuckDB 0.8.1 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rt
       n
   <dbl>
1 410683
```

## 4.5 Large scale characterisation

TO ADD

# 5 Adding cohorts to the CDM

## 5.1 What is a cohort?

When performing research with the OMOP common data model we often want to identify groups of individuals who share some set of characteristics. The criteria for including individuals can range from the seemingly simple (e.g. people diagnosed with asthma) to the much more complicated (e.g. adults diagnosed with asthma who had a year of prior observation time in the database prior to their diagnosis, had no prior history of chronic obstructive pulmonary disease, and no history of use of short-acting beta-antagonists).

The set of people we identify are cohorts, and the OMOP CDM has a specific structure by which they can be represented, with a cohort table having four required fields: 1) cohort definition id (a unique identifier for each cohort), 2) subject id (a foreign key to the subject in the cohort - typically referring to records in the person table), 3) cohort start date, and 4) cohort end date. Individuals can enter a cohort multiple times, but the time in which they are in the cohort cannot overlap.

## 5.2 Set up

```
library(CDMConnector)
library(dplyr)
library(PatientProfiles)
library(CodelistGenerator)
# library(IncidencePrevalence)
# library(DrugUtilisation)

db <- DBI::dbConnect(duckdb::duckdb(), eunomia_dir())

cdm <- cdm_from_con(
  con = db,
  cdm_schema = "main",
  write_schema = "main"
)
```

## 5.3 Creating a base cohort

### 5.3.1 General concept based cohort

```
cdm <- generate_concept_cohort_set(cdm,
                                   concept_set = list("gi_bleed" = 192671),
                                   limit = "all",
                                   end = 30,
                                   name = "gi_bleed",
                                   overwrite = TRUE)

cdm$gi_bleed %>%
  glimpse()
```

```
Rows: ??
Columns: 4
Database: DuckDB 0.8.1 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\RtmpY
$ cohort_definition_id <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~
$ subject_id           <int> 260, 549, 787, 795, 962, 1076, 1099, 1112, 1158, ~
$ cohort_start_date    <date> 2010-04-04, 1987-12-28, 2017-04-02, 1982-09-03, ~
$ cohort_end_date      <date> 2010-05-04, 1988-01-27, 2017-05-02, 1982-10-03, ~
```

### 5.3.2 Characteristic cohorts

We can use IncidencePrevalence ….

Here for example we'll generate a cohort of people aged between 18 and 50. Individuals will enter the cohort once they are in database and satisfy the age requirement.

```
# inc prev cohort
```

### 5.3.3 Drug-specific cohorts

Meanwhile, if we are interested in defining a drug cohort we can use the We can use `DrugUtilisation` package, where we have additional options around paremeters such as ……

```
# acetaminophen drug utilisation cohort
# instead of below
```

### Finding appropriate codes

In the defining the cohorts above we have needed to provide concept IDs to define our cohort. But, where do these come from?
We can search for codes of interest using the `CodelistGenerator` package. This can be done using a text search with the function `CodelistGenerator::getCandidateCodes()`. For example, we can find the GI code we use above like so:

```
getCandidateCodes(cdm = cdm,
                  keywords = "Gastrointestinal hemorrhage",
                  domains = "condition",
                  includeDescendants = TRUE)
```

```
# A tibble: 1 x 6
  concept_id concept_name    domain_id concept_class_id vocabulary_id found_from
       <int> <chr>           <chr>     <chr>            <chr>         <chr>
1     192671 Gastrointestin~ condition clinical finding snomed        From init~
```

We can also do automated searches that make use of the hierarchies in the vocabularies. Here, for example, we find the code for the drug ingredient Acetaminophen and all of it's descendants.

```
# TO ADD
```

Note that the data we're using just has a subset of the full OMOP CDM vocabularies. In practice, these searches would return many more codes. And in the case of the former in particular, clinical expertise would then be required to decide which of the codes were in line with the clinical idea at hand.

### Applying the appropriate logic when creating a cohort

As well as including appropriate concepts, we also face various other choices when defining our cohort. Decisions are required as to whether to include only the first event or the all events for an individual. We'll also need to make decisions cohort end dates, which could range from the same day as cohort entry to the end of an individual's observation period. For a drug utilisation cohort we then have even more decisions, such whether to combine cohorts with less than some specified days gap between one ending and the next starting.
These decisions for cohort logic will often reflect the way the cohort is being used in answering the study question. Like with including the right concepts, careful consideration will need to be taken when deciding on these parameters.

```
# To add
```

## 5.4 Cohort attributes

The set of cohorts we create will be associated with various attributes. The cohort set attribute contains information on the cohorts that we've generated.

```
cohortSet(cdm$gi_bleed) %>%
  glimpse()
```

```
Rows: 1
Columns: 2
$ cohort_definition_id <int> 1
$ cohort_name          <chr> "gi_bleed"
```

Another attribute contains counts of the cohorts we've created.

```
cohortCount(cdm$gi_bleed) %>%
  glimpse()
```

```
Rows: 1
Columns: 3
$ cohort_definition_id <int> 1
$ number_records       <dbl> 479
$ number_subjects      <dbl> 479
```

And we can also see attrition related to the cohort. We'll see below how any addition inclusion criteria that we apply can be recorded using this attrition attribute.

```
cohortAttrition(cdm$gi_bleed) %>%
  glimpse()
```

```
Rows: 1
Columns: 7
$ cohort_definition_id <int> 1
$ number_records       <dbl> 479
```

```
$ number_subjects      <dbl> 479
$ reason_id            <dbl> 1
$ reason               <chr> "Qualifying initial records"
$ excluded_records     <dbl> 0
$ excluded_subjects    <dbl> 0
```

## 5.5 Applying inclusion criteria

### 5.5.1 Applying demographic inclusion criteria

Say for our study we want to include people with a GI bleed who were aged 40 or over at the time. We can use the add variables with these characteristics as seen in chapter 4 and then filter accordingly. The function `CDMConnector::record_cohort_attrition()` will then update our cohort attributes as we can see below.

```
cdm$gi_bleed <- cdm$gi_bleed %>%
  addDemographics(indexDate = "cohort_start_date") %>%
  filter(age >= 40) %>%
  record_cohort_attrition("Age 18 or older") %>%
  filter(sex == "Male") %>%
  record_cohort_attrition("Male")
```

```
cohortCount(cdm$gi_bleed)
```

```
# A tibble: 1 x 3
  cohort_definition_id number_records number_subjects
                 <int>          <dbl>           <dbl>
1                    1             94              94
```

```
cohortAttrition(cdm$gi_bleed)
```

```
# A tibble: 3 x 7
  cohort_definition_id number_records number_subjects reason_id reason
                 <int>          <dbl>           <dbl>     <dbl> <chr>
1                    1            479             479         1 Qualifying init~
2                    1            183             183         2 Age 18 or older
3                    1             94              94         3 Male
# i 2 more variables: excluded_records <dbl>, excluded_subjects <dbl>
```

### 5.5.2 Applying cohort-based inclusion criteria

As well as requirements about specific demographics, we may also want to use another cohort for inclusion criteria. Let's say we want to exclude anyone with rheumatoid arthritis diagnosed before their GI bleed. We can first generate this cohort and then apply this additional exclusion criteria like so.

```
cdm <- generate_concept_cohort_set(cdm,
                            concept_set = list("acetaminophen" = c(1125315,
                                                                   1127078,
                                                                   1127433,
                                                                   40229134,
                                                                   40231925,
                                                                   40162522,
                                                                   19133768)),
                              limit = "all",
                            end = "event_end_date",
                            name = "acetaminophen",
                            overwrite = TRUE)

cdm$gi_bleed <- cdm$gi_bleed %>%
  addCohortIntersectFlag(targetCohortTable = "acetaminophen",
                         indexDate = "cohort_start_date",
                         window =c(-Inf, -1),
                         nameStyle = "acetaminophen_excl") %>%
  filter(acetaminophen_excl == 1) %>%
  record_cohort_attrition("Prior use of acetaminophen")
```

## 5.6 Creating multiple derived cohorts

Say we want a 3 gi bleed cohorts for 3 different age bands

TO BE DONE

We need to make a nice set of functions to do this first!

```
cdm <- generate_concept_cohort_set(cdm,
                            concept_set = list("gi_bleed" = 192671),
                            limit = "all",
                            end = 30,
                            name = "gi_bleed",
```

```
                              overwrite = TRUE)
cdm$gi_bleed <- cdm$gi_bleed %>%
  addDemographics(indexDate = "cohort_start_date")

# Not currently possible but would like something like the below to work
# where union also takes care of updating the ids of cohorts
# so we don't have any duplicates


# cdm$gi_bleed_strata <- union_all(
# cdm$gi_bleed %>%
#   filter(age >= 40) %>%
#   record_cohort_attrition("Age 18 or older"),
# cdm$gi_bleed %>%
#   filter(age <= 40) %>%
#   record_cohort_attrition("Age 18 or older"))
#
# cohortCount(cdm$gi_bleed_strata)
```

# 6 Working with cohorts

## 6.1 Cohort intersections

PatientProfiles::addCohortIntersect()

## 6.2 Intersection between two cohorts

## 6.3 Set up

```
library(CDMConnector)
library(dplyr)
library(PatientProfiles)

db <- DBI::dbConnect(duckdb::duckdb(), eunomia_dir())

cdm <- cdm_from_con(
  con = db,
  cdm_schema = "main",
  write_schema = "main"
)

cdm <- cdm %>%
  generate_concept_cohort_set(concept_set = list("gi_bleed" = 192671),
                              limit = "all",
                              end = 30,
                              name = "gi_bleed",
                              overwrite = TRUE) %>%
  generate_concept_cohort_set(concept_set = list("acetaminophen" = c(1125315,
                                                                     1127078,
                                                                     1127433,
                                                                     40229134,
                                                                     40231925,
```

```
                                                          40162522,
                                                          19133768)),
                            limit = "all",
                          # end = "event_end_date",
                          name = "acetaminophen",
                          overwrite = TRUE)
```

### 6.3.1 Flag

```
cdm$gi_bleed <- cdm$gi_bleed %>%
  addCohortIntersectFlag(targetCohortTable = "acetaminophen",
                         window = list(c(-Inf, -1), c(0,0), c(1, Inf)))

cdm$gi_bleed %>%
  summarise(acetaminophen_prior = sum(acetaminophen_minf_to_m1),
            acetaminophen_index = sum(acetaminophen_0_to_0),
            acetaminophen_post = sum(acetaminophen_1_to_inf)) %>%
  collect()
```

```
# A tibble: 1 x 3
  acetaminophen_prior acetaminophen_index acetaminophen_post
                <dbl>               <dbl>              <dbl>
1                 467                 467                476
```

### 6.3.2 Count

### 6.3.3 Date and times

## 6.4 Intersection between a cohort and tables with patient data

# 7 Summarising cohorts

## 7.1 Summarising patient demographics

PatientProfiles::summariseCharacteristics .....

## 7.2 Large scale characterisation

PatientProfiles::summariseLargeScaleCharacteristics .....

TO ADD

# 8 Comparing cohorts

# 9 Organising study code

# 10 Efficient study code

# References