Biostat 203B Homework 2

Due Feb 7, 2025 @ 11:59PM

Wenjing Zhou and 806542441

Display machine information for reproducibility:

```
sessionInfo()
```

```
R version 4.4.1 (2024-06-14)
Platform: x86_64-apple-darwin20
Running under: macOS 15.1

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.4-x86_64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.4-x86_64/Resources/lib/libRlapack.dylib;

locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

time zone: America/Los_Angeles
tzcode source: internal

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

loaded via a namespace (and not attached):
 [1] compiler_4.4.1    fastmap_1.2.0     cli_3.6.3         tools_4.4.1
 [5] htmltools_0.5.8.1 rstudioapi_0.17.0 yaml_2.3.10       rmarkdown_2.29
 [9] knitr_1.49        jsonlite_1.8.9    xfun_0.48         digest_0.6.37
[13] rlang_1.1.4       evaluate_1.0.1
```

Load necessary libraries (you can add more as needed).

```
library(arrow)
```

```
Attaching package: 'arrow'

The following object is masked from 'package:utils':

    timestamp
```

```
library(data.table)
library(duckdb)
```

```
Loading required package: DBI
```

```
library(memuse)
library(pryr)
```

```
Attaching package: 'pryr'

The following object is masked from 'package:data.table':

    address
```

```
library(R.utils)
```

```
Loading required package: R.oo

Loading required package: R.methodsS3

R.methodsS3 v1.8.2 (2022-06-13 22:00:14 UTC) successfully loaded. See ?R.methodsS3 for help.

R.oo v1.27.0 (2024-11-01 18:00:02 UTC) successfully loaded. See ?R.oo for help.

Attaching package: 'R.oo'
```

```
The following object is masked from 'package:R.methodsS3':

    throw

The following objects are masked from 'package:methods':

    getClasses, getMethods

The following objects are masked from 'package:base':

    attach, detach, load, save

R.utils v2.12.3 (2023-11-18 01:00:02 UTC) successfully loaded. See ?R.utils for help.


Attaching package: 'R.utils'

The following object is masked from 'package:arrow':

    timestamp

The following object is masked from 'package:utils':

    timestamp

The following objects are masked from 'package:base':

    cat, commandArgs, getOption, isOpen, nullfile, parse, use, warnings
```

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2
```

```
-- Conflicts --------------------------------------- tidyverse_conflicts() --
x dplyr::between()      masks data.table::between()
x purrr::compose()      masks pryr::compose()
x lubridate::duration() masks arrow::duration()
x tidyr::extract()      masks R.utils::extract()
x dplyr::filter()       masks stats::filter()
x dplyr::first()        masks data.table::first()
x lubridate::hour()     masks data.table::hour()
x lubridate::isoweek()  masks data.table::isoweek()
x dplyr::lag()          masks stats::lag()
x dplyr::last()         masks data.table::last()
x lubridate::mday()     masks data.table::mday()
x lubridate::minute()   masks data.table::minute()
x lubridate::month()    masks data.table::month()
x purrr::partial()      masks pryr::partial()
x lubridate::quarter()  masks data.table::quarter()
x lubridate::second()   masks data.table::second()
x purrr::transpose()    masks data.table::transpose()
x lubridate::wday()     masks data.table::wday()
x lubridate::week()     masks data.table::week()
x dplyr::where()        masks pryr::where()
x lubridate::yday()     masks data.table::yday()
x lubridate::year()     masks data.table::year()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becor
```

Display memory information of your computer

```
memuse::Sys.meminfo()
```

```
Totalram:  16.000 GiB
Freeram:    3.668 GiB
```

In this exercise, we explore various tools for ingesting the MIMIC-IV data introduced in homework 1.

Display the contents of MIMIC `hosp` and `icu` data folders:

```
ls -l ~/mimic/hosp/
```

```
total 11185032
-rw-r--r--@ 1 vickey  staff    19928140 Jun 24  2024 admissions.csv.gz
```

```
-rw-r--r--@ 1 vickey  staff      427554 Apr 12  2024 d_hcpcs.csv.gz
-rw-r--r--@ 1 vickey  staff      876360 Apr 12  2024 d_icd_diagnoses.csv.gz
-rw-r--r--@ 1 vickey  staff      589186 Apr 12  2024 d_icd_procedures.csv.gz
-rw-r--r--@ 1 vickey  staff       13169 Oct  3 09:07 d_labitems.csv.gz
-rw-r--r--@ 1 vickey  staff    33564802 Oct  3 09:07 diagnoses_icd.csv.gz
-rw-r--r--@ 1 vickey  staff     9743908 Oct  3 09:07 drgcodes.csv.gz
-rw-r--r--@ 1 vickey  staff   811305629 Apr 12  2024 emar.csv.gz
-rw-r--r--@ 1 vickey  staff     2162335 Apr 12  2024 hcpcsevents.csv.gz
-rw-r--r--@ 1 vickey  staff  2592909134 Oct  3 09:08 labevents.csv.gz
-rw-r--r--@ 1 vickey  staff   174094381 Feb  7 15:55 labevents_filtered.csv.gz
-rw-r--r--@ 1 vickey  staff   117644075 Oct  3 09:08 microbiologyevents.csv.gz
-rw-r--r--@ 1 vickey  staff    44069351 Oct  3 09:08 omr.csv.gz
-rw-r--r--@ 1 vickey  staff     2835586 Apr 12  2024 patients.csv.gz
-rw-r--r--@ 1 vickey  staff   525708076 Apr 12  2024 pharmacy.csv.gz
-rw-r--r--@ 1 vickey  staff   666594177 Apr 12  2024 poe.csv.gz
-rw-r--r--@ 1 vickey  staff    55267894 Apr 12  2024 poe_detail.csv.gz
-rw-r--r--@ 1 vickey  staff   606298611 Apr 12  2024 prescriptions.csv.gz
-rw-r--r--@ 1 vickey  staff     7777324 Apr 12  2024 procedures_icd.csv.gz
-rw-r--r--@ 1 vickey  staff      127330 Apr 12  2024 provider.csv.gz
-rw-r--r--@ 1 vickey  staff     8569241 Apr 12  2024 services.csv.gz
-rw-r--r--@ 1 vickey  staff    46185771 Oct  3 09:08 transfers.csv.gz
```

```
ls -l ~/mimic/icu/
```

```
total 8506784
-rw-r--r--@ 1 vickey  staff       41566 Apr 12  2024 caregiver.csv.gz
-rw-r--r--@ 1 vickey  staff  3502392765 Apr 12  2024 chartevents.csv.gz
-rw-r--r--@ 1 vickey  staff       58741 Apr 12  2024 d_items.csv.gz
-rw-r--r--@ 1 vickey  staff    63481196 Apr 12  2024 datetimeevents.csv.gz
-rw-r--r--@ 1 vickey  staff     3342355 Oct  3 07:36 icustays.csv.gz
-rw-r--r--@ 1 vickey  staff   311642048 Apr 12  2024 ingredientevents.csv.gz
-rw-r--r--@ 1 vickey  staff   401088206 Apr 12  2024 inputevents.csv.gz
-rw-r--r--@ 1 vickey  staff    49307639 Apr 12  2024 outputevents.csv.gz
-rw-r--r--@ 1 vickey  staff    24096834 Apr 12  2024 procedureevents.csv.gz
```

Q1. `read.csv` (base R) vs `read_csv` (tidyverse) vs `fread` (data.table)

Q1.1 Speed, memory, and data types

There are quite a few utilities in R for reading plain text data files. Let us test the speed of reading a moderate sized compressed csv file, `admissions.csv.gz`, by three functions: `read.csv` in base R, `read_csv` in tidyverse, and `fread` in the data.table package.

Which function is fastest? Is there difference in the (default) parsed data types? How much memory does each resultant dataframe or tibble use? (Hint: `system.time` measures run times; `pryr::object_size` measures memory usage; all these readers can take gz file as input without explicit decompression.)

```
setwd("~/mimic/hosp/")

#speed
# Using read.csv (base R)
system.time({
  admissions_base <- read.csv(gzfile("admissions.csv.gz"))
})
```

```
   user  system elapsed
  9.577   0.192   9.793
```

```
# Using read_csv (readr in the tidyverse)
library(readr)
system.time({
  admissions_tidy <- read_csv("admissions.csv.gz")
})
```

```
Rows: 546028 Columns: 16
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (8): admission_type, admit_provider_id, admission_location, discharge_l...
dbl  (3): subject_id, hadm_id, hospital_expire_flag
dttm (5): admittime, dischtime, deathtime, edregtime, edouttime

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
   user  system elapsed
  2.723   0.308   1.282
```

```
# Using fread (data.table)
library(data.table)
system.time({
  admissions_dt <- fread("admissions.csv.gz")
})
```

```
   user  system elapsed
  1.537   0.123   0.657
```

```
object_size(admissions_base)
```

```
200.10 MB
```

```
object_size(admissions_tidy)
```

```
70.02 MB
```

```
object_size(admissions_dt)
```

```
63.47 MB
```

Solution: Function `fread` is the fastest way to read the data. The data types are different in these three functions. `read.cvs` returns a data frame. `read_csv` returns a tibble. `fread` returns a data table. The memory usage of `fread` is the smallest, while `read.csv` takes the most memory.

Q1.2 User-supplied data types

Re-ingest `admissions.csv.gz` by indicating appropriate column data types in `read_csv`. Does the run time change? How much memory does the result tibble use? (Hint: `col_types` argument in `read_csv`.)

```
con <- gzfile("~/mimic/hosp/admissions.csv.gz", "rt")
data <- read.csv(con)
close(con)

head(data)
```

```
  subject_id  hadm_id           admittime           dischtime deathtime
1   10000032 22595853 2180-05-06 22:23:00 2180-05-07 17:15:00
2   10000032 22841357 2180-06-26 18:27:00 2180-06-27 18:49:00
3   10000032 25742920 2180-08-05 23:44:00 2180-08-07 17:50:00
4   10000032 29079034 2180-07-23 12:35:00 2180-07-25 17:55:00
5   10000068 25022803 2160-03-03 23:16:00 2160-03-04 06:26:00
```

```
6   10000084 23052089 2160-11-21 01:56:00 2160-11-25 14:52:00
  admission_type admit_provider_id     admission_location discharge_location
1         URGENT            P49AFC TRANSFER FROM HOSPITAL                HOME
2       EW EMER.            P784FA         EMERGENCY ROOM                HOME
3       EW EMER.            P19UTS         EMERGENCY ROOM             HOSPICE
4       EW EMER.            P06OTX         EMERGENCY ROOM                HOME
5 EU OBSERVATION            P39NWO         EMERGENCY ROOM
6       EW EMER.            P42H7G  WALK-IN/SELF REFERRAL   HOME HEALTH CARE
  insurance language marital_status  race           edregtime
1  Medicaid  English        WIDOWED WHITE 2180-05-06 19:17:00
2  Medicaid  English        WIDOWED WHITE 2180-06-26 15:54:00
3  Medicaid  English        WIDOWED WHITE 2180-08-05 20:58:00
4  Medicaid  English        WIDOWED WHITE 2180-07-23 05:54:00
5            English         SINGLE WHITE 2160-03-03 21:55:00
6  Medicare  English        MARRIED WHITE 2160-11-20 20:36:00
            edouttime hospital_expire_flag
1 2180-05-06 23:30:00                    0
2 2180-06-26 21:31:00                    0
3 2180-08-06 01:44:00                    0
4 2180-07-23 14:00:00                    0
5 2160-03-04 06:26:00                    0
6 2160-11-21 03:20:00                    0
```

```
setwd("~/mimic/hosp/")

col_spec <- cols(
  subject_id = col_integer(),
  hadm_id = col_integer(),
  admittime = col_datetime(format = "%Y-%m-%d %H:%M:%S"),
  dischtime = col_datetime(format = "%Y-%m-%d %H:%M:%S"),
  deathtime = col_datetime(format = "%Y-%m-%d %H:%M:%S"),
  admission_type = col_character(),
  admit_provider_id = col_character(),
  admission_location = col_character(),
  discharge_location = col_character(),
  insurance = col_character(),
  language = col_character(),
  marital_status = col_character(),
  race = col_character(),
  edregtime = col_datetime(format = "%Y-%m-%d %H:%M:%S"),
  edouttime = col_datetime(format = "%Y-%m-%d %H:%M:%S"),
  hospital_expire_flag = col_integer(),
```

```
)


time_spec <- system.time({
  admissions_spec <- read_csv("admissions.csv.gz", col_types = col_spec)
})
print(time_spec)
```

```
   user  system elapsed
  2.343   0.307   0.935
```

```
memory_usage <- pryr::object_size(admissions_spec)
print(memory_usage)
```

```
63.47 MB
```

Solution: The run time is similar to the previous one. The memory usage is smaller than the previous one which is 63.47 MB.

Q2. Ingest big data files

Let us focus on a bigger file, `labevents.csv.gz`, which is about 130x bigger than `admissions.csv.gz`.

```
ls -l ~/mimic/hosp/labevents.csv.gz
```

```
-rw-r--r--@ 1 vickey  staff  2592909134 Oct  3 09:08 /Users/vickey/mimic/hosp/labevents.csv.g
```

Display the first 10 lines of this file.

```
zcat < ~/mimic/hosp/labevents.csv.gz | head -10
```

```
labevent_id,subject_id,hadm_id,specimen_id,itemid,order_provider_id,charttime,storetime,valu
1,10000032,,2704548,50931,P69FQC,2180-03-23 11:51:00,2180-03-23 15:56:00,___,95,mg/dL,70,100
2,10000032,,36092842,51071,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,ROUTINE,
3,10000032,,36092842,51074,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,ROUTINE,
4,10000032,,36092842,51075,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,ROUTINE,"
5,10000032,,36092842,51079,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,ROUTINE,
6,10000032,,36092842,51087,P69FQC,2180-03-23 11:51:00,,,,,,,ROUTINE,RANDOM.
7,10000032,,36092842,51089,P69FQC,2180-03-23 11:51:00,2180-03-23 16:15:00,,,,,,ROUTINE,PRESU
8,10000032,,36092842,51090,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,ROUTINE,M
9,10000032,,36092842,51092,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,ROUTINE,"
```

Q2.1 Ingest `labevents.csv.gz` by `read_csv`

Try to ingest `labevents.csv.gz` using `read_csv`. What happens? If it takes more than 3 minutes on your computer, then abort the program and report your findings.

```
system.time({
  lab_events <- read_csv("~/mimic/hosp/labevents.csv.gz")
})
```

Solution: It takes more than 3 minutes to read the data. This confirms that `read_csv` is not a good choice for big data files.

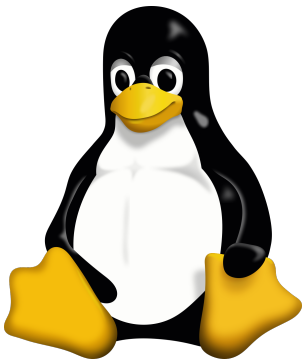
Q2.2 Ingest selected columns of `labevents.csv.gz` by `read_csv`

Try to ingest only columns `subject_id`, `itemid`, `charttime`, and `valuenum` in `labevents.csv.gz` using `read_csv`. Does this solve the ingestion issue? (Hint: `col_select` argument in `read_csv`.)

```
system.time({
  lab_events_subset <- read_csv("~/mimic/hosp/labevents.csv.gz",
                                col_select = c("subject_id", "itemid",
                                               "charttime", "valuenum"))
})
```

Solution: It still takes about seven minutes to read the data.

Q2.3 Ingest a subset of `labevents.csv.gz`



Our first strategy to handle this big data file is to make a subset of the `labevents` data. Read the MIMIC documentation for the content in data file `labevents.csv`.

In later exercises, we will only be interested in the following lab items: creatinine (50912), potassium (50971), sodium (50983), chloride (50902), bicarbonate (50882), hematocrit (51221), white blood cell count (51301), and glucose (50931) and the following columns: `subject_id`, `itemid`, `charttime`, `valuenum`. Write a Bash command to extract these columns and rows from `labevents.csv.gz` and save the result to a new file `labevents_filtered.csv.gz` in the current working directory. (Hint: Use `zcat <` to pipe the output of `labevents.csv.gz` to `awk` and then to `gzip` to compress the output. Do **not** put `labevents_filtered.csv.gz` in Git! To save render time, you can put `#| eval: false` at the beginning of this code chunk. TA will change it to `#| eval: true` before rendering your qmd file.)

```
zcat < ~/mimic/hosp/labevents.csv.gz| awk -F',' 'BEGIN { OFS="," }
NR==1 {
  # Identify the column numbers for the desired columns
  for (i = 1; i <= NF; i++) {
    if ($i == "subject_id") subj = i;
    else if ($i == "itemid") item = i;
    else if ($i == "charttime") time = i;
    else if ($i == "valuenum") val = i;
```

```
  }
  print $subj, $item, $time, $val;
  next;
}
{
  # Check if itemid is one of the desired lab items
  if ($item == "50912" || $item == "50971" || $item == "50983" ||
      $item == "50902" || $item == "50882" || $item == "51221" ||
      $item == "51301" || $item == "50931")
    print $subj, $item, $time, $val;
}' | gzip > ~/mimic/hosp/labevents_filtered.csv.gz
```

Display the first 10 lines of the new file `labevents_filtered.csv.gz`. How many lines are in this new file, excluding the header? How long does it take `read_csv` to ingest `labevents_filtered.csv.gz`?

```
system("zcat<~/mimic/hosp/labevents_filtered.csv.gz | head -10")
```

```
time_taken <- system.time({
  labevents <- read_csv("~/mimic/hosp/labevents_filtered.csv.gz")
})
```

```
Rows: 32679896 Columns: 4
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl  (3): subject_id, itemid, valuenum
dttm (1): charttime

i Use `spec()` to retrieve the full column specification for this data.
i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
print(time_taken)
```

```
   user  system elapsed
 46.326   3.763  12.093
```

Solution: There are 32679895 lines in this new file, excluding the header. It takes about 14.748 seconds to read the data.

Q2.4 Ingest `labevents.csv` by Apache Arrow

Our second strategy is to use Apache Arrow for larger-than-memory data analytics. Unfortunately Arrow does not work with gz files directly. First decompress `labevents.csv.gz` to `labevents.csv` and put it in the current working directory (do not add it in git!). To save render time, put `#| eval: false` at the beginning of this code chunk. TA will change it to `#| eval: true` when rendering your qmd file.

Then use `arrow::open_dataset` to ingest `labevents.csv`, select columns, and filter `itemid` as in Q2.3. How long does the ingest+select+filter process take? Display the number of rows and the first 10 rows of the result tibble, and make sure they match those in Q2.3. (Hint: use `dplyr` verbs for selecting columns and filtering rows.)

```
gzip -dk < ~/mimic/hosp/labevents.csv.gz > ./labevents.csv
```

```
start_time <- Sys.time()

ds <- open_dataset("labevents.csv", format = "csv")

result <- ds %>%
  select(subject_id, itemid, charttime, valuenum) %>%
  filter(itemid %in% c(50912, 50971, 50983, 50902, 50882, 51221, 51301,
                       50931)) %>%
  collect()  # Pull the filtered data into an R tibble

end_time <- Sys.time()
time_taken <- end_time - start_time


print(time_taken)
```

```
Time difference of 40.26228 secs
```

```
# Display the number of data rows (excluding the header)
cat("Number of rows (excluding header):", nrow(result), "\n")
```

```
Number of rows (excluding header): 32679896
```

```
# Display the first 10 rows of the result tibble
print(head(result, 10))
```

```
# A tibble: 10 x 4
   subject_id itemid charttime           valuenum
        <int>  <int> <dttm>                 <dbl>
 1   10000032  50931 2180-03-23 04:51:00     95
 2   10000032  50882 2180-03-23 04:51:00     27
 3   10000032  50902 2180-03-23 04:51:00    101
 4   10000032  50912 2180-03-23 04:51:00      0.4
 5   10000032  50971 2180-03-23 04:51:00      3.7
 6   10000032  50983 2180-03-23 04:51:00    136
 7   10000032  51221 2180-03-23 04:51:00     45.4
 8   10000032  51301 2180-03-23 04:51:00      3
 9   10000032  51221 2180-05-06 15:25:00     42.6
10   10000032  51301 2180-05-06 15:25:00      5
```

Write a few sentences to explain what is Apache Arrow. Imagine you want to explain it to a layman in an elevator.

Solution: Apache Arrow is a cross-language development platform for in-memory data. It specifies a standardized language-independent columnar memory format for flat and hierarchical data, organized for efficient analytic operations on modern hardware like CPUs and GPUs. Arrow also provides libraries for many programming languages to manipulate this data format, enabling zero-copy reads for lightning-fast data access without serialization overhead. Think of it as a way to store and move my data around without needing to copy or reformat it each time—saving my time and making my computer work smarter, not harder.

Q2.5 Compress `labevents.csv` to Parquet format and ingest/select/filter

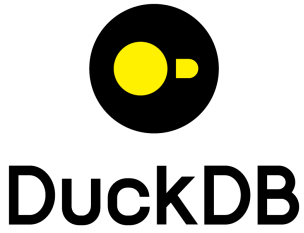
Re-write the csv file `labevents.csv` in the binary Parquet format (Hint: `arrow::write_dataset`.) How large is the Parquet file(s)? How long does the ingest+select+filter process of the Parquet file(s) take? Display the number of rows and the first 10 rows of the result tibble and make sure they match those in Q2.3. (Hint: use `dplyr` verbs for selecting columns and filtering rows.)

```r
ds_csv <- open_dataset("labevents.csv", format = "csv")

# Write the dataset in Parquet format into a directory
write_dataset(ds_csv, path = "labevents_parquet", format = "parquet")

start_time <- Sys.time()

ds_parquet <- open_dataset("labevents_parquet", format = "parquet")

result_parquet <- ds_parquet %>%
  select(subject_id, itemid, charttime, valuenum) %>%
  filter(itemid %in% c(50912, 50971, 50983, 50902, 50882, 51221, 51301, 50931)) %>%
  collect()

end_time <- Sys.time()
time_taken <- end_time - start_time
print(time_taken)


cat("Number of rows (excluding header):", nrow(result_parquet), "\n")

print(head(result_parquet, 10))
```

Write a few sentences to explain what is the Parquet format. Imagine you want to explain it to a layman in an elevator.

Solution: Parquet format is a columnar storage file format optimized for use with big data processing frameworks. It is designed to bring efficiency compared to row-based files like CSV, especially for complex nested data structures. Parquet files are highly efficient in terms of both storage and processing speed, making them ideal for large-scale data analytics tasks. They achieve this efficiency through techniques like columnar storage, compression, and encoding schemes.

Q2.6 DuckDB



Ingest the Parquet file, convert it to a DuckDB table by `arrow::to_duckdb`, select columns, and filter rows as in Q2.5. How long does the ingest+convert+select+filter process take? Display the number of rows and the first 10 rows of the result tibble and make sure they match those in Q2.3. (Hint: use `dplyr` verbs for selecting columns and filtering rows.)

```
parquet_dir <- "labevents_parquet"

start_time <- Sys.time()

ds <- open_dataset(parquet_dir, format = "parquet")

duck_tbl <- to_duckdb(ds)

result <- duck_tbl %>%
  select(subject_id, hadm_id, itemid, charttime, valuenum, value, flag) %>%
  filter(itemid %in% c(50912, 50971, 50983, 50902, 50882, 51221, 51301, 50931)) %>%
  collect()

end_time <- Sys.time()
time_taken <- end_time - start_time

cat("Ingest + convert + select + filter process took:", time_taken, "seconds\n\n")
```

```
Ingest + convert + select + filter process took: 13.54808 seconds
```

```
cat("Number of rows in the filtered result:", nrow(result), "\n\n")
```

```
Number of rows in the filtered result: 32679896
```

```
cat("First 10 rows of the result:\n")
```

```
First 10 rows of the result:
```

```
print(head(result, 10))
```

```
# A tibble: 10 x 7
   subject_id  hadm_id itemid charttime           valuenum value flag
        <dbl>    <dbl>  <dbl> <dttm>                 <dbl> <chr> <chr>
 1   10000980       NA  51301 2191-05-23 10:20:00      4.6 4.6   ""
 2   10000980 25911675  50882 2191-05-24 05:45:00     25   25    ""
```

```
 3   10000980 25911675  50902 2191-05-24 05:45:00    108   108   ""
 4   10000980 25911675  50912 2191-05-24 05:45:00      2.1 2.1   "abnormal"
 5   10000980 25911675  50931 2191-05-24 05:45:00    116   ___   "abnormal"
 6   10000980 25911675  50971 2191-05-24 05:45:00      4   4.0   ""
 7   10000980 25911675  50983 2191-05-24 05:45:00    144   144   ""
 8   10000980 25911675  51221 2191-05-24 05:45:00     28   28.0  "abnormal"
 9   10000980 25911675  51301 2191-05-24 05:45:00      3.4 3.4   "abnormal"
10   10000980       NA  51221 2191-05-30 12:40:00     28.8 28.8  "abnormal"
```

Write a few sentences to explain what is DuckDB. Imagine you want to explain it to a layman in an elevator.

Solution: DuckDB is an in-process SQL database designed specifically for fast analytical queries. In practical terms, it's like having a high-speed, mini data warehouse on your laptop that lets you run complex queries without setting up a separate server. This makes it perfect for data scientists and analysts who need to explore and analyze huge datasets on the fly, all within a familiar SQL interface.

Q3. Ingest and filter `chartevents.csv.gz`

chartevents.csv.gz contains all the charted data available for a patient. During their ICU stay, the primary repository of a patient's information is their electronic chart. The `itemid` variable indicates a single measurement type in the database. The `value` variable is the value measured for `itemid`. The first 10 lines of `chartevents.csv.gz` are

```
zcat < ~/mimic/icu/chartevents.csv.gz | head -10
```

```
subject_id,hadm_id,stay_id,caregiver_id,charttime,storetime,itemid,value,valuenum,valueuom,wa
10000032,29079034,39553978,18704,2180-07-23 12:36:00,2180-07-23 14:45:00,226512,39.4,39.4,kg
10000032,29079034,39553978,18704,2180-07-23 12:36:00,2180-07-23 14:45:00,226707,60,60,Inch,0
10000032,29079034,39553978,18704,2180-07-23 12:36:00,2180-07-23 14:45:00,226730,152,152,cm,0
10000032,29079034,39553978,18704,2180-07-23 14:00:00,2180-07-23 14:18:00,220048,SR (Sinus Rhy
10000032,29079034,39553978,18704,2180-07-23 14:00:00,2180-07-23 14:18:00,224642,Oral,,,0
10000032,29079034,39553978,18704,2180-07-23 14:00:00,2180-07-23 14:18:00,224650,None,,,0
10000032,29079034,39553978,18704,2180-07-23 14:00:00,2180-07-23 14:20:00,223761,98.7,98.7,°F
10000032,29079034,39553978,18704,2180-07-23 14:11:00,2180-07-23 14:17:00,220179,84,84,mmHg,0
10000032,29079034,39553978,18704,2180-07-23 14:11:00,2180-07-23 14:17:00,220180,48,48,mmHg,0
```

How many rows? 433 millions. 432997491 rows.

```
zcat < ~/mimic/icu/chartevents.csv.gz | tail -n +2 | wc -l
```

432997491 rows.

d_items.csv.gz is the dictionary for the `itemid` in `chartevents.csv.gz`.

```
zcat < ~/mimic/icu/d_items.csv.gz | head -10
```

```
itemid,label,abbreviation,linksto,category,unitname,param_type,lownormalvalue,highnormalvalue
220001,Problem List,Problem List,chartevents,General,,Text,,
220003,ICU Admission date,ICU Admission date,datetimeevents,ADT,,Date and time,,
220045,Heart Rate,HR,chartevents,Routine Vital Signs,bpm,Numeric,,
220046,Heart rate Alarm - High,HR Alarm - High,chartevents,Alarms,bpm,Numeric,,
220047,Heart Rate Alarm - Low,HR Alarm - Low,chartevents,Alarms,bpm,Numeric,,
220048,Heart Rhythm,Heart Rhythm,chartevents,Routine Vital Signs,,Text,,
220050,Arterial Blood Pressure systolic,ABPs,chartevents,Routine Vital Signs,mmHg,Numeric,90
220051,Arterial Blood Pressure diastolic,ABPd,chartevents,Routine Vital Signs,mmHg,Numeric,60
220052,Arterial Blood Pressure mean,ABPm,chartevents,Routine Vital Signs,mmHg,Numeric,,
```

In later exercises, we are interested in the vitals for ICU patients: heart rate (220045), mean non-invasive blood pressure (220181), systolic non-invasive blood pressure (220179), body temperature in Fahrenheit (223761), and respiratory rate (220210). Retrieve a subset of `chartevents.csv.gz` only containing these items, using the favorite method you learnt in Q2.

Document the steps and show code. Display the number of rows and the first 10 rows of the result tibble.

```
# Define the path to the chartevents CSV file (gzipped)
chartevents_file <- "~/mimic/icu/chartevents.csv.gz"

# Ingest the file using Arrow
ds <- open_dataset(chartevents_file, format = "csv")

# Convert the Arrow dataset into a DuckDB table
duck_tbl <- to_duckdb(ds)

# the list of itemids for the vitals of interest
vitals_itemids <- c(220045, 220181, 220179, 223761, 220210)

# Filter the DuckDB table for only those rows and select all columns
result <- duck_tbl %>%
```

```
  filter(itemid %in% vitals_itemids) %>%
  collect()

cat("Number of rows in the subset:", nrow(result), "\n\n")
```

```
Number of rows in the subset: 30195426
```

```
# first 10 rows of the result
cat("First 10 rows:\n")
```

```
First 10 rows:
```

```
print(head(result, 10))
```

```
# A tibble: 10 x 11
   subject_id  hadm_id  stay_id caregiver_id charttime
        <dbl>    <dbl>    <dbl>        <dbl> <dttm>
 1   10003400 20214994 32128372        70054 2137-02-26 16:00:00
 2   10003400 20214994 32128372        70054 2137-02-26 16:00:00
 3   10003400 20214994 32128372        70054 2137-02-26 16:00:00
 4   10003400 20214994 32128372        70054 2137-02-26 16:00:00
 5   10003400 20214994 32128372        70054 2137-02-26 16:00:00
 6   10003400 20214994 32128372        70054 2137-02-26 17:00:00
 7   10003400 20214994 32128372        70054 2137-02-26 17:00:00
 8   10003400 20214994 32128372        70054 2137-02-26 17:00:00
 9   10003400 20214994 32128372        70054 2137-02-26 17:00:00
10   10003400 20214994 32128372        70054 2137-02-26 18:00:00
# i 6 more variables: storetime <dttm>, itemid <dbl>, value <chr>,
#   valuenum <dbl>, valueuom <chr>, warning <dbl>
```

Solution: I selected DuckDB as my favourite method. The number of rows in the subset is 30195426.