Biostat 203B Homework 2

Due Feb 7, 2025 @ 11:59PM

Khoa Vu UID: 705600710

Display machine information for reproducibility:

```
sessionInfo()
```

```
R version 4.4.2 (2024-10-31)
Platform: x86_64-pc-linux-gnu
Running under: Ubuntu 24.04.1 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.12.0
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.12.0

locale:
 [1] LC_CTYPE=C.UTF-8       LC_NUMERIC=C           LC_TIME=C.UTF-8
 [4] LC_COLLATE=C.UTF-8     LC_MONETARY=C.UTF-8    LC_MESSAGES=C.UTF-8
 [7] LC_PAPER=C.UTF-8       LC_NAME=C              LC_ADDRESS=C
[10] LC_TELEPHONE=C         LC_MEASUREMENT=C.UTF-8 LC_IDENTIFICATION=C

time zone: America/Los_Angeles
tzcode source: system (glibc)

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

loaded via a namespace (and not attached):
 [1] compiler_4.4.2    fastmap_1.2.0     cli_3.6.3         tools_4.4.2
 [5] htmltools_0.5.8.1 rstudioapi_0.17.1 yaml_2.3.10       rmarkdown_2.29
 [9] knitr_1.49        jsonlite_1.8.9    xfun_0.50         digest_0.6.37
[13] rlang_1.1.4       evaluate_1.0.3
```

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 1.9.4     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:  9.717 GiB
Freeram:   8.804 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 24124664
-rwxrwxrwx 1 kvu1702 kvu1702    19928140 Jan 14 21:13 admissions.csv.gz
```

```
-rwxrwxrwx 1 kvu1702 kvu1702      427554 Jan 14 21:13 d_hcpcs.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702      876360 Jan 14 21:13 d_icd_diagnoses.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702      589186 Jan 14 21:13 d_icd_procedures.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702       13169 Jan 14 21:13 d_labitems.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702    33564802 Jan 14 21:13 diagnoses_icd.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702     9743908 Jan 14 21:13 drgcodes.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702   811305629 Jan 14 21:13 emar.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702   748158322 Jan 14 21:13 emar_detail.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702     2162335 Jan 14 21:13 hcpcsevents.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702        2907 Jan 14 21:13 index.html
-rwxrwxrwx 1 kvu1702 kvu1702 18402851720 Jan 14 21:13 labevents.csv
-rwxrwxrwx 1 kvu1702 kvu1702  2592909134 Jan 14 21:13 labevents.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702   117644075 Jan 14 21:14 microbiologyevents.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702    44069351 Jan 14 21:14 omr.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702     2835586 Jan 14 21:14 patients.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702   525708076 Jan 14 21:14 pharmacy.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702   666594177 Jan 14 21:14 poe.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702    55267894 Jan 14 21:14 poe_detail.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702   606298611 Jan 14 21:14 prescriptions.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702     7777324 Jan 14 21:14 procedures_icd.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702      127330 Jan 14 21:14 provider.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702     8569241 Jan 14 21:14 services.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702    46185771 Jan 14 21:14 transfers.csv.gz
```

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

```
total 4253396
-rwxrwxrwx 1 kvu1702 kvu1702      41566 Jan 14 21:14 caregiver.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702 3502392765 Jan 14 21:14 chartevents.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702      58741 Jan 14 21:15 d_items.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702   63481196 Jan 14 21:15 datetimeevents.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702    3342355 Jan 14 21:15 icustays.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702       1336 Jan 14 21:15 index.html
-rwxrwxrwx 1 kvu1702 kvu1702  311642048 Jan 14 21:15 ingredientevents.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702  401088206 Jan 14 21:15 inputevents.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702   49307639 Jan 14 21:15 outputevents.csv.gz
-rwxrwxrwx 1 kvu1702 kvu1702   24096834 Jan 14 21:15 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.)

Solution:

Testing the speed of `read.csv`:

```
system.time(read.csv("~/mimic/hosp/admissions.csv.gz"))
```

```
   user  system elapsed
 10.941   0.197  11.883
```

```
pryr::object_size(read.csv("~/mimic/hosp/admissions.csv.gz"))
```

```
200.10 MB
```

Testing the speed of `read_csv`:

```
system.time(read_csv("~/mimic/hosp/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
  3.342   0.515   2.233
```

```
pryr::object_size(read_csv("~/mimic/hosp/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.
```

```
70.02 MB
```

Testing the speed of `fread`:

```
system.time(fread("~/mimic/hosp/admissions.csv.gz"))
```

```
   user  system elapsed
  1.585   0.090   1.303
```

```
pryr::object_size(fread("~/mimic/hosp/admissions.csv.gz"))
```

```
63.47 MB
```

In order of decreasing speed, read.csv() was the slowest, taking ~10 seconds, followed by read_csv(), taking ~1.8 seconds. fread() was the fastest, taking ~1.3 seconds. In order of decreasing memory usage, read.csv() was the largest, taking ~200 MB, followed by read_csv(), taking up ~70 MB. fread() took the least amount of memory at ~63.5 MB.

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`.)

Solution:

```
#Note: we can use a compact string representation where each character
#represents one column:
#c = character
#i = integer
#n = number
#d = double
#l = logical
#f = factor
#D = date
#T = date time
#t = time
#? = guess
#From read_csv:
#subject_id --> INTEGER NOT NULL --> integer (i)
#hadm_id --> INTEGER NOT NULL --> integer (i)
#admittime --> TIMESTAMP NOT NULL --> data time (T)
#dischtime --> TIMESTAMP --> data time (T)
#deathtime --> TIMESTAMP --> data time (T)
#admission_type --> VARCHAR(40) NOT NULL --> character (c)
#admit_provider_id --> VARCHAR(10) --> character (c)
#admission_location --> VARCHAR(60) --> character (c)
#discharge_location --> VARCHAR(60) --> character (C)
#insurance --> VARCHAR(255) --> character (C)
#language --> VARCHAR(10) --> character (C)
#marital_status --> VARCHAR(30) --> character (C)
#race --> VARCHAR(80) --> character (C)
#edregtime --> TIMESTAMP --> data time (T)
#edouttime --> TIMESTAMP --> data time (T)
#hospital_expire_flag --> SMALLINT --> integer (i)
colum_data_types <- c("i", "i", "T", "T", "T", "i", "c", "c", "c",
                      "c", "c", "c", "c", "T", "T", "i")
system.time(read_csv("~/mimic/hosp/admissions.csv.gz",
                     col_types = colum_data_types))
```

```
   user  system elapsed
  1.553   0.241   1.521
```

```
pryr::object_size(read_csv("~/mimic/hosp/admissions.csv.gz",
                     col_types = colum_data_types))
```

```
67.84 MB
```

When indicating the appropriate column types, the runtime increases to ~1.9 seconds, and the memory used increases to ~68 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
```

```
-rwxrwxrwx 1 kvu1702 kvu1702 2592909134 Jan 14 21:13 /home/kvu1702/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,value
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,MI
9,10000032,,36092842,51092,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,ROUTINE,"C
```

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.

Solution:

```
system.time(read_csv("~/mimic/hosp/labevents.csv.gz"))
pryr::object_size(read_csv("~/mimic/hosp/labevents.csv.gz"))
```

When trying to use `read_csv` to ingest `labevents.csv.gz`, my computer crashes. This is most likely due to a lack of processing power, available memory, etc. on my laptop.

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`.)

Solution:

```
column_subset <- c("subject_id", "itemid", "charttime", "valuenum")
system.time(read_csv("~/mimic/hosp/labevents.csv.gz",
                     col_select = column_subset))
pryr::object_size(read_csv("~/mimic/hosp/labevents.csv.gz",
                     col_select = column_subset))
```

When trying to use `read_csv` to ingest only columns `subject_id`, `itemid`, `charttime`, and `valuenum` in `labevents.csv.gz`, my computer crashes still crashes, despite reading only a subset.

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.)

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`?

Solution:

```
#We use awk to select our columns of interest.
#`subject_id`, `itemid`, `charttime`, `valuenum` are columns 2, 5, 7, and 10
#respectively and to select values of the following lab items:
#creatinine (50912), potassium (50971), sodium (50983), chloride (50902),
```

```
#bicarbonate (50882), hematocrit (51221), white blood cell count (51301),
#and glucose (50931) using itemid, column 5.

#Note: We use NR == 1 to skip the logic in the first line, since we want to
#keep the headers to read later on.
zcat < ~/mimic/hosp/labevents.csv.gz |
awk -F ',' 'NR == 1 || $5 ~ /50912|50971|50983|50902|50882|51221|51301|50931/ \
{print $2 "," $5 "," $7 "," $10}' |  gzip > ~/labevents_filtered.csv.gz
```

```
echo 'The first ten lines in labevents_filtered.csv.gz are:'
zcat < ~/labevents_filtered.csv.gz | head -n 10

#To count the rows without the header, we pipe tail -n +2
echo 'The line number, minus the header, in labevents_filtered.csv.gz is:'
zcat < ~/labevents_filtered.csv.gz | tail -n +2 |wc -l
```

```
The first ten lines in labevents_filtered.csv.gz are:
subject_id,itemid,charttime,valuenum
10000032,50931,2180-03-23 11:51:00,95
10000032,50882,2180-03-23 11:51:00,27
10000032,50902,2180-03-23 11:51:00,101
10000032,50912,2180-03-23 11:51:00,0.4
10000032,50971,2180-03-23 11:51:00,3.7
10000032,50983,2180-03-23 11:51:00,136
10000032,51221,2180-03-23 11:51:00,45.4
10000032,51301,2180-03-23 11:51:00,3
10000032,51221,2180-05-06 22:25:00,42.6
The line number, minus the header, in labevents_filtered.csv.gz is:
32679896
```

```
system.time(fread("~/labevents_filtered.csv.gz"))
```

```
   user  system elapsed
  7.404   1.461   5.448
```

```
pryr::object_size(fread("~/labevents_filtered.csv.gz"))
```

```
784.32 MB
```

The number of lines in `labevents_filtered.csv`, minus the header, is 32679896 Reading the filtered dataset takes ~9 seconds and ~784 MB of memory.

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.)

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

Solution:

```
#We first unzip labevents.csv.gz
gzip -dk ~/mimic/hosp/labevents.csv.gz
```

```
subset_columns <- c("subject_id", "itemid", "charttime", "valuenum")
subset_itemid <-c(50912, 50971, 50983, 50902,
                  50882, 51221, 51301, 50931)

system.time(
  arrow::open_dataset("~/mimic/hosp/labevents.csv", format = "csv") %>%
  select(all_of(subset_columns)) %>%
  filter(itemid %in% subset_itemid) %>%
  arrange(subject_id, charttime, itemid) %>%
  collect()
  )
```

```
   user  system elapsed
 45.435   6.376 152.925
```

```
df <- arrow::open_dataset("~/mimic/hosp/labevents.csv", format = "csv") %>%
  select(all_of(subset_columns)) %>%
  filter(itemid %in% subset_itemid) %>%
  arrange(subject_id, charttime, itemid) %>%
  collect()

nrow(df)
```

```
[1] 32679896
```

```
head(df, 10)
```

```
# A tibble: 10 x 4
   subject_id itemid charttime           valuenum
        <int>  <int> <dttm>                 <dbl>
 1   10000032  50882 2180-03-23 04:51:00     27
 2   10000032  50902 2180-03-23 04:51:00    101
 3   10000032  50912 2180-03-23 04:51:00      0.4
 4   10000032  50931 2180-03-23 04:51:00     95
 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  50882 2180-05-06 15:25:00     27
10   10000032  50902 2180-05-06 15:25:00    105
```

```
#Clearing the df variable to save on memory
rm(list = ls())
gc()
```

```
          used (Mb) gc trigger  (Mb)  max used   (Mb)
Ncells 1832352 97.9    5047575 269.6   7886835  421.3
Vcells 5275410 40.3  130562580 996.2 136383971 1040.6
```

It takes ~2 minutes for the ingestion + selecting + filtering + sorting of labevents.csv

Apache Arrow is a software development platform built for high-performance applications involved in transporting and processing large data sets. Its in-memory columnar format holds language-independent specifications to structure table-like datasetsets. Apache Arrow has libraries implemented in various languages, including C, C++, Java, MATLAB, Python, R, and Julia.

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.)

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

Solution:

```
#Writing the Parquet file
df <- arrow::open_dataset("~/mimic/hosp/labevents.csv", format = "csv")
arrow::write_dataset(df, path = "~/labevents_pq", format = "parquet")
#Clearing the df variable to save on memory
rm(list = ls())
gc()
```

```
         used  (Mb) gc trigger  (Mb)  max used   (Mb)
Ncells 1913907 102.3    5047575 269.6   7886835  421.3
Vcells 5418954  41.4  104450064 796.9 136383971 1040.6
```

```
#Reading the data set
subset_columns <- c("subject_id", "itemid", "charttime", "valuenum")
subset_itemid <-c(50912, 50971, 50983, 50902,
                  50882, 51221, 51301, 50931)

system.time(
  arrow::open_dataset("~/labevents_pq/part-0.parquet") %>%
  select(all_of(subset_columns)) %>%
  filter(itemid %in% subset_itemid) %>%
  arrange(subject_id, charttime, itemid) %>%
  collect()
  )
```

```
   user  system elapsed
 25.649   4.470   9.940
```

```
df <- arrow::open_dataset("~/labevents_pq/part-0.parquet") %>%
  select(all_of(subset_columns)) %>%
  filter(itemid %in% subset_itemid) %>%
  arrange(subject_id, charttime, itemid) %>%
  collect()

nrow(df)
```

```
[1] 32679896
```

```
head(df, 10)
```

```
# A tibble: 10 x 4
   subject_id itemid charttime           valuenum
        <int>  <int> <dttm>                 <dbl>
 1   10000032  50882 2180-03-23 04:51:00     27
 2   10000032  50902 2180-03-23 04:51:00    101
 3   10000032  50912 2180-03-23 04:51:00      0.4
 4   10000032  50931 2180-03-23 04:51:00     95
 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  50882 2180-05-06 15:25:00     27
10   10000032  50902 2180-05-06 15:25:00    105
```

```
#Clearing the df variable to save on memory
rm(list = ls())
gc()
```

```
         used  (Mb) gc trigger   (Mb)  max used   (Mb)
Ncells 1913929 102.3    5047575  269.6   7886835  421.3
Vcells 5419031  41.4  135054860 1030.4 168818575 1288.0
```

Ingesting, selecting, filtering, and sorting the Parquet file took ~17 seconds. The Parquet file is ~2.5 GB.

Optimized to handle flat columnar storage data formats, the Parquet open-source file format is very compatible with large-volume, complex data. Able to handle many encoding types, the Parquet file format is also known for its exemplary data compression ability. Using Google's record shredding, Parquet files can perform fast queries that select specific columns without the need to read the entire data set and perform efficient column-wise compression.

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.)

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

Solution:

```
subset_columns <- c("subject_id", "itemid", "charttime", "valuenum")
subset_itemid <-c(50912, 50971, 50983, 50902,
                  50882, 51221, 51301, 50931)

system.time(arrow::open_dataset("~/labevents_pq/part-0.parquet"))
```

```
   user  system elapsed
   0.40    0.01    0.41
```

```
df <- arrow::open_dataset("~/labevents_pq/part-0.parquet")

system.time(
  arrow::to_duckdb(df) %>%
  select(all_of(subset_columns)) %>%
  filter(itemid %in% subset_itemid) %>%
  arrange(subject_id, charttime, itemid) %>%
  collect()
)
```

```
   user  system elapsed
 62.012 107.854  67.161
```

```
df_duckdb <- arrow::to_duckdb(df) %>%
  select(all_of(subset_columns)) %>%
  filter(itemid %in% subset_itemid) %>%
  arrange(subject_id, charttime, itemid) %>%
  collect()

nrow(df_duckdb)
```

```
[1] 32679896
```

```
head(df_duckdb, 10)
```

```
# A tibble: 10 x 4
   subject_id itemid charttime           valuenum
        <dbl>  <dbl> <dttm>                 <dbl>
 1   10000032  50882 2180-03-23 11:51:00     27
 2   10000032  50902 2180-03-23 11:51:00    101
 3   10000032  50912 2180-03-23 11:51:00      0.4
 4   10000032  50931 2180-03-23 11:51:00     95
 5   10000032  50971 2180-03-23 11:51:00      3.7
 6   10000032  50983 2180-03-23 11:51:00    136
 7   10000032  51221 2180-03-23 11:51:00     45.4
 8   10000032  51301 2180-03-23 11:51:00      3
 9   10000032  50882 2180-05-06 22:25:00     27
10   10000032  50902 2180-05-06 22:25:00    105
```

```
#Clearing the df variable to save on memory
rm(list = ls())
gc()
```

```
          used  (Mb) gc trigger   (Mb)  max used   (Mb)
Ncells 2045705 109.3    5047575  269.6   7886835  421.3
Vcells 5676239  43.4  132615018 1011.8 168818575 1288.0
```

Solution: In total, it takes ~0.3 seconds to ingest the Parquet file and an extra ~1 minute to convert, select, filter, and sort the DuckDB file. In total, the whole process took ~6.3 seconds.

DuckDB is a portable, analytical, in-process, and open-source database system. DuckDB uses a rich SQL dialect to read and write files in many supported formats and perform lightning-fast queries using its columnar engine, which supports parallel execution. Unlike other database systems, DuckDB is easy to install and runs in-process in many different host applications, such as Rstudio.

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.

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

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.

Solution:

We use the methodology of Q2.6, converting `chartevents.csv.gz` first to a parquet file and then a DuckDB table before ingesting, filtering, and sorting.

```r
#Writing the Parquet file
df <- arrow::open_dataset("~/mimic/icu/chartevents.csv.gz", format = "csv")
arrow::write_dataset(df, path = "~/chartevents_pq", format = "parquet")
#Clearing the df variable to save on memory
rm(list = ls())
gc()
```

```
         used  (Mb) gc trigger  (Mb)  max used   (Mb)
Ncells 2044488 109.2    5047575 269.6   7886835  421.3
Vcells 5675786  43.4  106092015 809.5 168818575 1288.0
```

```r
subset_columns <- c("subject_id", "itemid", "charttime", "valuenum")
subset_itemid <-c(220045, 220181, 220179, 223761, 220210)
df <- arrow::open_dataset("~/chartevents_pq/part-0.parquet")
df_duckdb <- arrow::to_duckdb(df) %>%
  select(all_of(subset_columns)) %>%
  filter(itemid %in% subset_itemid) %>%
  arrange(subject_id, charttime, itemid) %>%
  collect()

nrow(df_duckdb)
```

```
[1] 30195426
```

```r
head(df_duckdb, 10)
```

```
# A tibble: 10 x 4
   subject_id itemid charttime           valuenum
        <dbl>  <dbl> <dttm>                 <dbl>
 1   10000032 223761 2180-07-23 14:00:00     98.7
 2   10000032 220179 2180-07-23 14:11:00     84
 3   10000032 220181 2180-07-23 14:11:00     56
 4   10000032 220045 2180-07-23 14:12:00     91
 5   10000032 220210 2180-07-23 14:12:00     24
 6   10000032 220045 2180-07-23 14:30:00     93
 7   10000032 220179 2180-07-23 14:30:00     95
 8   10000032 220181 2180-07-23 14:30:00     67
 9   10000032 220210 2180-07-23 14:30:00     21
10   10000032 220045 2180-07-23 15:00:00     94
```

```r
#Clearing variables to save on ram
rm(list = ls())
gc()
```

```
          used  (Mb) gc trigger  (Mb)  max used   (Mb)
Ncells 2045258 109.3    5047575 269.6   7886835  421.3
Vcells 5676570  43.4  123074959 939.0 168818575 1288.0
```

The number of rows is 30195426.