Biostat 203B Homework 2

Due Feb 7, 2025 @ 11:59PM

AUTHOR
Tanya Wang, 605587605

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_1.1.5       evaluate_1.0.3
```

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

```
library(arrow)
```

```
Some features are not enabled in this build of Arrow. Run `arrow_info()` for more information.


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 ──
✓ dplyr     1.1.4     ✓ readr     2.1.5
✓ forcats   1.0.0     ✓ stringr   1.5.1
✓ ggplot2   3.5.1     ✓ tibble    3.2.1
✓ lubridate 1.9.4     ✓ tidyr     1.3.1
✓ purrr     1.0.2
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::between()      masks data.table::between()
✖ purrr::compose()      masks pryr::compose()
✖ lubridate::duration() masks arrow::duration()
✖ tidyr::extract()      masks R.utils::extract()
✖ dplyr::filter()       masks stats::filter()
✖ dplyr::first()        masks data.table::first()
✖ lubridate::hour()     masks data.table::hour()
✖ lubridate::isoweek()  masks data.table::isoweek()
✖ dplyr::lag()          masks stats::lag()
✖ dplyr::last()         masks data.table::last()
✖ lubridate::mday()     masks data.table::mday()
✖ lubridate::minute()   masks data.table::minute()
✖ lubridate::month()    masks data.table::month()
✖ purrr::partial()      masks pryr::partial()
✖ lubridate::quarter()  masks data.table::quarter()
✖ lubridate::second()   masks data.table::second()
✖ purrr::transpose()    masks data.table::transpose()
✖ lubridate::wday()     masks data.table::wday()
✖ lubridate::week()     masks data.table::week()
✖ dplyr::where()        masks pryr::where()
✖ lubridate::yday()     masks data.table::yday()
✖ lubridate::year()     masks data.table::year()
ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become
errors
```

Display memory information of your computer

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

```
Totalram:  15.426 GiB
Freeram:    9.124 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 6323188
-rwxrwxrwx 1 tttanyaw tttanyaw   19928140 Feb  2 21:16 admissions.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw     427554 Feb  2 21:16 d_hcpcs.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw     876360 Feb  2 21:16 d_icd_diagnoses.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw     589186 Feb  2 21:16 d_icd_procedures.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw      13169 Feb  2 21:16 d_labitems.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw   33564802 Feb  2 21:16 diagnoses_icd.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw    9743908 Feb  2 21:16 drgcodes.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw  811305629 Feb  2 21:16 emar.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw  748158322 Feb  2 21:16 emar_detail.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw    2162335 Feb  2 21:16 hcpcsevents.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw 2592909134 Feb  2 21:16 labevents.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw  174144176 Feb  4 00:34 labevents_filtered.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw  117644075 Feb  2 21:16 microbiologyevents.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw   44069351 Feb  2 21:16 omr.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw    2835586 Feb  2 21:16 patients.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw  525708076 Feb  2 21:16 pharmacy.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw  666594177 Feb  2 21:16 poe.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw   55267894 Feb  2 21:16 poe_detail.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw  606298611 Feb  2 21:17 prescriptions.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw    7777324 Feb  2 21:17 procedures_icd.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw     127330 Feb  2 21:17 provider.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw    8569241 Feb  2 21:17 services.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw   46185771 Feb  2 21:17 transfers.csv.gz
```

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

```
total 4253392
-rwxrwxrwx 1 tttanyaw tttanyaw      41566 Feb  2 21:17 caregiver.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw 3502392765 Feb  2 21:17 chartevents.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw          0 Feb  4 00:07 chartevents_filtered.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw      58741 Feb  2 21:17 d_items.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw   63481196 Feb  2 21:17 datetimeevents.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw    3342355 Feb  2 21:17 icustays.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw  311642048 Feb  2 21:17 ingredientevents.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw  401088206 Feb  2 21:17 inputevents.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw   49307639 Feb  2 21:17 outputevents.csv.gz
-rwxrwxrwx 1 tttanyaw tttanyaw   24096834 Feb  2 21:17 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:

Test the speed of reading file:

```
# Define file path
file_path_admis <- "/home/tttanyaw/mimic/hosp/admissions.csv.gz"

# Read using base R
system.time(df_base <- read.csv(file_path_admis))
```

```
   user  system elapsed 
  6.916   0.129   7.306 
```

```
# Read using tidyverse
system.time(df_tidy <- read_csv(file_path_admis))
```

```
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 
  0.736   0.211   0.854 
```

```
# Read using data.table
system.time(df_dt <- fread(file_path_admis))
```

```
   user  system elapsed 
  1.039   0.101   0.637 
```

According to the results, fread() (Data.table) is the fastest function.

Data types:

```
sapply(df_base, class)   # Base R
```

```
          subject_id             hadm_id           admittime
           "integer"           "integer"         "character"
           dischtime           deathtime      admission_type
         "character"         "character"         "character"
   admit_provider_id admission_location discharge_location
         "character"         "character"         "character"
           insurance            language      marital_status
         "character"         "character"         "character"
                race           edregtime           edouttime
         "character"         "character"         "character"
hospital_expire_flag
           "integer"
```

```
sapply(df_tidy, class)   # Tidyverse
```

```
$subject_id
[1] "numeric"

$hadm_id
[1] "numeric"

$admittime
[1] "POSIXct" "POSIXt"

$dischtime
[1] "POSIXct" "POSIXt"

$deathtime
[1] "POSIXct" "POSIXt"

$admission_type
[1] "character"

$admit_provider_id
[1] "character"

$admission_location
[1] "character"

$discharge_location
[1] "character"

$insurance
[1] "character"

$language
[1] "character"

$marital_status
[1] "character"
```

```
$race
[1] "character"

$edregtime
[1] "POSIXct" "POSIXt"

$edouttime
[1] "POSIXct" "POSIXt"

$hospital_expire_flag
[1] "numeric"
```

```
sapply(df_dt, class)     # data.table
```

```
$subject_id
[1] "integer"

$hadm_id
[1] "integer"

$admittime
[1] "POSIXct" "POSIXt"

$dischtime
[1] "POSIXct" "POSIXt"

$deathtime
[1] "POSIXct" "POSIXt"

$admission_type
[1] "character"

$admit_provider_id
[1] "character"

$admission_location
[1] "character"

$discharge_location
[1] "character"

$insurance
[1] "character"

$language
[1] "character"

$marital_status
[1] "character"
```

```
$race
[1] "character"

$edregtime
[1] "POSIXct" "POSIXt"

$edouttime
[1] "POSIXct" "POSIXt"

$hospital_expire_flag
[1] "integer"
```

Therefore, there are differences in the default parsed data types.

Check the momory:

```
# read.csv() (Base R)
object_size(df_base)
```

```
200.10 MB
```

```
# read_csv() (Tidyverse)
object_size(df_tidy)
```

```
70.02 MB
```

```
# fread() (Data.table)
object_size(df_dt)
```

```
63.47 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:

```
# Indicate appropriate column data types
appro_data_type <- cols(
  subject_id = col_integer(),
  hadm_id = col_integer(),
  admittime = col_datetime(format = ""),
  dischtime = col_datetime(format = ""),
  deathtime = col_datetime(format = ""),
  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 = ""),
  edouttime = col_datetime(format = ""),
  hospital_expire_flag = col_integer()
)

# Run time
system.time(df_tidy_appro <- read_csv(file_path_admis,
                                      col_types = appro_data_type))
```

```
   user  system elapsed 
  0.487   0.162   0.698 
```

```
# Memory
object_size(df_tidy)
```

```
70.02 MB
```

```
object_size(df_tidy_appro)
```

```
63.47 MB
```

According to the output results, specifying col_types results in a longer run time, and it uses 63.47 MB of momory, which is less than the 70.02 MB used by the default read_csv() auto-detection.

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 tttanyaw tttanyaw 2592909134 Feb  2 21:16 /home/tttanyaw/mimic/hosp/labevents.csv.gz
```

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,val
uenum,valueuom,ref_range_lower,ref_range_upper,flag,priority,comments
1,10000032,,2704548,50931,P69FQC,2180-03-23 11:51:00,2180-03-23
15:56:00,___,95,mg/dL,70,100,,ROUTINE,"IF FASTING, 70-100 NORMAL, >125 PROVISIONAL DIABETES."
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,"BENZODIAZEPINE IMMUNOASSAY SCREEN DOES NOT DETECT SOME
DRUGS,;INCLUDING LORAZEPAM, CLONAZEPAM, AND FLUNITRAZEPAM."
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,PRESUMPTIVELY POSITIVE.
8,10000032,,36092842,51090,P69FQC,2180-03-23 11:51:00,2180-03-23
16:00:00,NEG,,,,,,ROUTINE,METHADONE ASSAY DETECTS ONLY METHADONE (NOT OTHER OPIATES/OPIOIDS).
9,10000032,,36092842,51092,P69FQC,2180-03-23 11:51:00,2180-03-23
16:00:00,NEG,,,,,,ROUTINE,"OPIATE IMMUNOASSAY SCREEN DOES NOT DETECT SYNTHETIC OPIOIDS;SUCH AS
METHADONE, OXYCODONE, FENTANYL, BUPRENORPHINE, TRAMADOL,;NALOXONE, MEPERIDINE.  SEE ONLINE LAB
MANUAL FOR DETAILS."
```

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:

```
file_path_lab <- "/home/tttanyaw/mimic/hosp/labevents.csv.gz"
system.time(df_labevents <- read_csv("file_path_lab"))
```

It took more than 3 minutes on my computer and the RStudio is not responding after that. I used `free -h` to monitor memory usage in Ubuntu and found that 15 out of 15 GB memory was used, meaning that my RAM is almost fully used.

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:

```
col_select <- c("subject_id", "itemid", "charttime", "valuenum")
system.time(df_selected <- read_csv(file_path_lab, col_select = col_select))
```

No, it still takes more than 3 minutes to run.

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:

```
zcat < ~/mimic/hosp/labevents.csv.gz | head -1
```

```
labevent_id,subject_id,hadm_id,specimen_id,itemid,order_provider_id,charttime,storetime,value,val
uenum,valueuom,ref_range_lower,ref_range_upper,flag,priority,comments
```

```
zcat < ~/mimic/hosp/labevents.csv.gz |
awk -F',' 'NR==1 || $5 ~ /50912|50971|50983|50902|50882|51221|51301|50931/' |
cut -d',' -f2,5,7,10 |
gzip > ~/mimic/hosp/labevents_filtered.csv.gz
```

Display the first 10 lines:

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

```
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
```

Count the number of lines excluding the header:

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

```
32679896
```

Time for `read_csv` to ingest `labevents_filtered.csv.gz`:

```
system.time(df_filtered <- read_csv("labevents_filtered.csv.gz"))
```

```
Rows: 42875 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.

   user  system elapsed
  0.015   0.017   0.034
```

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:

Decompress `labevents.csv.gz`:

```
gunzip -c ~/mimic/hosp/labevents.csv.gz > labevents.csv
echo "labevents.csv" >> .gitignore
```

Ingest + select + filter:

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

# select columns and filter
filtered_labevents <- labevents %>%
  select(subject_id, itemid, charttime, valuenum) %>%
  filter(itemid %in% c(50912, 50971, 50983, 50902, 50882, 51221, 51301, 50931))
```

```
# measure time
system.time(df_arrow <- collect(filtered_labevents))
```

```
   user  system elapsed 
 22.977   7.471  20.150 
```

Number of rows:

```
nrow(df_arrow)
```

```
[1] 32679896
```

First 10 rows:

```
head(df_arrow, 10)
```

```
# A tibble: 10 × 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 
```

Apache Arrow is a high-performance framework designed for fast and efficient data processing. Compare to traditional methods that require data to be read and written repeatedly, Apache Arrow keeps data in a columnar memory format, allowing seamless interaction between different analysis tools. This will prevent unnecessary data conversion and reduces processing time and memory usage. By optimizing how data is stored and accessed, Arrow enables real-time analysis of large data sets that would otherwise be too slow or memory intensive to process.For example, it is like a online video platform that does not need people to download movies, instead, they can access to the data without waiting.

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:

Convert `labevents.csv` to binary Parquet Format:

```
write_dataset(
  open_dataset("labevents.csv", format = "csv"),
  path = "labevents_parquet",
  format = "parquet"
)
```

Check Parquet File Size:

```
ls -lh labevents_parquet/
```

```
total 2.6G
-rw-r--r-- 1 tttanyaw tttanyaw 2.6G Feb  4 00:44 part-0.parquet
```

Ingest+select+filter process:

```
data_parquet <- open_dataset("labevents_parquet", format = "parquet")

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

system.time(df_parquet <- collect(filtered_data_parquet))
```

```
   user  system elapsed 
  9.215   5.370   4.267 
```

Number of rows:

```
nrow(df_parquet)
```

```
[1] 32679896
```

First 10 rows:

```
head(df_parquet, 10)
```

```
# A tibble: 10 × 4
   subject_id itemid charttime           valuenum
        <int>  <int> <dttm>                 <dbl>
 1   10002155  50971 2129-08-14 23:25:00      5.4
 2   10002155  50983 2129-08-14 23:25:00    128  
 3   10002155  51221 2129-08-14 23:25:00     29.4
 4   10002155  51301 2129-08-14 23:25:00      5.8
 5   10002155  50882 2129-08-15 08:20:00     25  
```

```
 6   10002155  50902 2129-08-15 08:20:00     95
 7   10002155  50912 2129-08-15 08:20:00      1.1
 8   10002155  50931 2129-08-15 08:20:00     98
 9   10002155  50971 2129-08-15 08:20:00      5
10   10002155  50983 2129-08-15 08:20:00    128
```

Parquet is like a ZIP file for big data. Instead of storing data in long, slow text files like CSV, Parquet organizes it efficiently in a compressed, column-based format. This makes reading data much faster and reduces storage space. For example, CSV is like a library where books are scattered randoml, but Parquet is like a well-organized library with a digital index.

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:

Ingest + convert + select + filter:

```
data_parquet <- open_dataset("labevents_parquet", format = "parquet")
duckdb_table <- to_duckdb(data_parquet)

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

# Measure execution time
system.time(df_duckdb <- collect(filtered_data_duckdb))
```

```
   user  system elapsed
 12.827   4.442   4.019
```

Number of rows:

```
nrow(df_duckdb)
```

```
[1] 32679896
```

First 10 rows:

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

```
# A tibble: 10 × 4
   subject_id itemid charttime           valuenum
        <dbl>  <dbl> <dttm>                 <dbl>
 1   10007920  50971 2139-01-01 07:37:00      4.1
 2   10007920  50983 2139-01-01 07:37:00    141
 3   10007920  51221 2139-01-07 11:38:00     42.7
 4   10007920  51301 2139-01-07 11:38:00      8.1
 5   10007920  51221 2139-06-24 11:24:00     37.2
 6   10007920  51301 2139-06-24 11:24:00      6.7
 7   10007920  50912 2139-06-24 11:24:00      0.9
 8   10007920  51221 2139-12-30 12:30:00     40.7
 9   10007920  51301 2139-12-30 12:30:00      9.1
10   10007920  50912 2139-12-30 12:30:00      1.5
```

DuckDB lets you run super-fast, SQL-like queries on large datasets, but without needing a huge database server. Unlike traditional databases that are slow for analysis, DuckDB is lightweight, in-memory, and optimized for analytics. For example, it is like online shared website that can handle millions of rows instantly instead of crashing.

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,warnin
g
10000032,29079034,39553978,18704,2180-07-23 12:36:00,2180-07-23 14:45:00,226512,39.4,39.4,kg,0
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
Rhythm),,,0
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,0
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,90,140
220051,Arterial Blood Pressure diastolic,ABPd,chartevents,Routine Vital Signs,mmHg,Numeric,60,90
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:

Decompress `chartevents.csv.gz`:

```
gunzip -c ~/mimic/icu/chartevents.csv.gz > chartevents.csv
echo "chartevents.csv" >> .gitignore
```

Ingest + select + filter:

```
chartevents <- open_dataset(
  "chartevents.csv",
  format = "csv"
)

# select columns and filter
filtered_charteventss <- chartevents %>%
  select(subject_id, itemid, charttime, valuenum) %>%
  filter(itemid %in% c(220045, 220181, 220179, 223761, 220210))

# measure time
system.time(df_arrow_new <- collect(filtered_charteventss))
```

```
   user  system elapsed 
 50.322  15.482  45.443 
```

Number of rows:

```
nrow(df_arrow_new)
```

```
[1] 30195426
```

First 10 rows:

```
head(df_arrow_new, 10)
```

```
# A tibble: 10 × 4
   subject_id itemid charttime           valuenum
        <int>  <int> <dttm>                 <dbl>
 1   10000032 223761 2180-07-23 07:00:00     98.7
 2   10000032 220179 2180-07-23 07:11:00     84
 3   10000032 220181 2180-07-23 07:11:00     56
 4   10000032 220045 2180-07-23 07:12:00     91
 5   10000032 220210 2180-07-23 07:12:00     24
 6   10000032 220045 2180-07-23 07:30:00     93
 7   10000032 220179 2180-07-23 07:30:00     95
 8   10000032 220181 2180-07-23 07:30:00     67
 9   10000032 220210 2180-07-23 07:30:00     21
10   10000032 220045 2180-07-23 08:00:00     94
```