Biostat 203B Homework 2

Due Feb 7, 2025 @ 11:59PM

Wenqiang Ge UID:106371961

Table of contents

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)
```

```

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 beco
```

Display memory information of your computer

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

```
Totalram:  7.686 GiB 
Freeram:   6.750 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 6153128
-rwxrwxrwx 1 gewenqiang gewenqiang   19928140 Jan 16 19:00 admissions.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang     427554 Jan 16 19:00 d_hcpcs.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang     876360 Jan 16 19:00 d_icd_diagnoses.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang     589186 Jan 16 19:00 d_icd_procedures.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang      13169 Jan 16 19:00 d_labitems.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang   33564802 Jan 16 19:00 diagnoses_icd.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang    9743908 Jan 16 19:00 drgcodes.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang  811305629 Jan 16 19:00 emar.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang  748158322 Jan 16 19:00 emar_detail.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang    2162335 Jan 16 19:00 hcpcsevents.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang       2907 Jan 16 19:00 index.html
-rwxrwxrwx 1 gewenqiang gewenqiang 2592909134 Jan 16 19:01 labevents.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang  117644075 Jan 16 19:01 microbiologyevents.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang   44069351 Jan 16 19:01 omr.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang    2835586 Jan 16 19:01 patients.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang  525708076 Jan 16 19:01 pharmacy.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang  666594177 Jan 16 19:01 poe.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang   55267894 Jan 16 19:01 poe_detail.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang  606298611 Jan 16 19:01 prescriptions.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang    7777324 Jan 16 19:01 procedures_icd.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang     127330 Jan 16 19:01 provider.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang    8569241 Jan 16 19:01 services.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang   46185771 Jan 16 19:01 transfers.csv.gz
```

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

```
total 4253396
-rwxrwxrwx 1 gewenqiang gewenqiang      41566 Jan 16 19:01 caregiver.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang 3502392765 Jan 16 19:02 chartevents.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang      58741 Jan 16 19:02 d_items.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang   63481196 Jan 16 19:02 datetimeevents.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang    3342355 Jan 16 19:02 icustays.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang       1336 Jan 16 19:02 index.html
-rwxrwxrwx 1 gewenqiang gewenqiang  311642048 Jan 16 19:02 ingredientevents.csv.gz
```

```
-rwxrwxrwx 1 gewenqiang gewenqiang  401088206 Jan 16 19:02 inputevents.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang   49307639 Jan 16 19:02 outputevents.csv.gz
-rwxrwxrwx 1 gewenqiang gewenqiang   24096834 Jan 16 19:02 procedureevents.csv.gz
```

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

0.1.1 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 data frame 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:

```
library(microbenchmark)
library(readr)
library(data.table)

base = read.csv("~/mimic/hosp/admissions.csv.gz")
readr = read_csv("~/mimic/hosp/admissions.csv.gz", show_col_types = FALSE)
dt = fread("~/mimic/hosp/admissions.csv.gz")

benchmark_results <- microbenchmark(
  base , readr , dt , times = 5
)
print(benchmark_results)
```

```
Unit: nanoseconds
  expr min lq  mean median uq  max neval
  base  40 40 302.4     40 60 1332     5
 readr  30 40 358.8     61 90 1573     5
    dt  40 40 988.0     70 91 4699     5
```

```
rm(benchmark_results)
```

```
library(pryr)
size_base <- object_size(base)
size_readr <- object_size(readr)
size_fread <- object_size(dt)
```

```
print(data.frame(
  Method = c("Base R", "readr", "data.table"),
  Memory_MB = c(size_base, size_readr, size_fread)
))
```

```
      Method Memory_MB
1     Base R 200.10 MB
2      readr  70.02 MB
3 data.table  63.47 MB
```

The fastest function of reading a compressed csv file is fread. The best memory efficiency function is fread().

read.csv() (Base R): Converts character columns to factors.

read_csv() (readr): Keeps character columns as character.

fread() (data.table): Automatically detects optimal types for speed & memory efficiency.

0.1.2 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:

```
file_path <- "~/mimic/hosp/admissions.csv.gz"

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

```
system.time(df_optimized <- read_csv(file_path, col_types = col_spec))
```

```
   user  system elapsed 
  1.460   0.462   1.357 
```

```
memory_optimized <- object_size(df_optimized) / (1024^2)  # Convert to MB
print(paste("Optimized Tibble Memory Usage:",
            round(memory_optimized, 2), "MB"))
```

```
[1] "Optimized Tibble Memory Usage: 60.53 MB"
```

Yes, both the runtime and memory have changed. The running time has become shorter and the required memory for operation has also decreased.

0.2 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 gewenqiang gewenqiang 2592909134 Jan 16 19:01 /home/gewenqiang/mimic/hosp/labev
```

Display the first 10 lines of this file.

Solution:

```
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,70,10
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,PRES
8,10000032,,36092842,51090,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,,ROUTINE,N
9,10000032,,36092842,51092,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,,ROUTINE,'
```

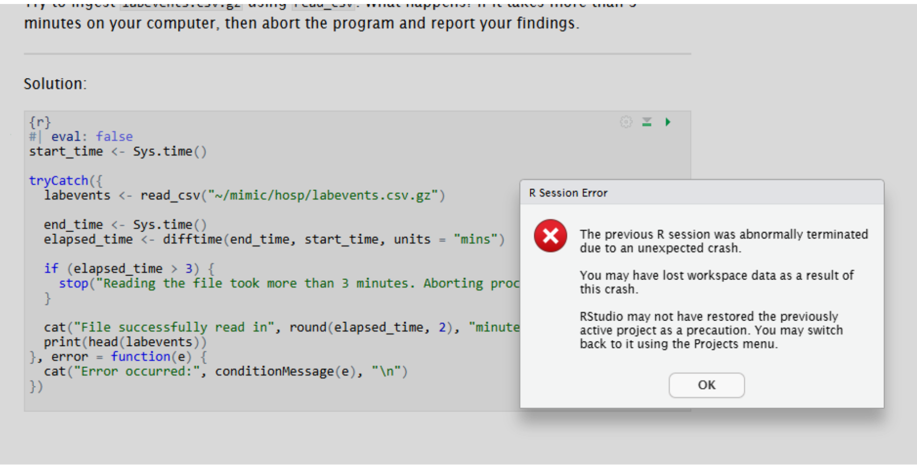
0.2.1 Q2.1 Ingest `labevents.csv.gz` by `read_csv`

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:

```
start_time <- Sys.time()
```

```
tryCatch({
  labevents <- read_csv("~/mimic/hosp/labevents.csv.gz")

  end_time <- Sys.time()
  elapsed_time <- difftime(end_time, start_time, units = "mins")

  if (elapsed_time > 3) {
    stop("Reading the file took more than 3 minutes. Aborting process.")
  }

  cat("File successfully read in", round(elapsed_time, 2), "minutes.\n")
  print(head(labevents))
}, error = function(e) {
  cat("Error occurred:", conditionMessage(e), "\n")
})
```



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:

```
{r}
#| eval: false
start_time <- Sys.time()

tryCatch({
  labevents <- read_csv("~/mimic/hosp/labevents.csv.gz")

  end_time <- Sys.time()
  elapsed_time <- difftime(end_time, start_time, units = "mins")

  if (elapsed_time > 3) {
    stop("Reading the file took more than 3 minutes. Aborting proc
  }

  cat("File successfully read in", round(elapsed_time, 2), "minute
  print(head(labevents))
}, error = function(e) {
  cat("Error occurred:", conditionMessage(e), "\n")
})
```

R Session Error

The previous R session was abnormally terminated due to an unexpected crash.

You may have lost workspace data as a result of this crash.

RStudio may not have restored the previously active project as a precaution. You may switch back to it using the Projects menu.

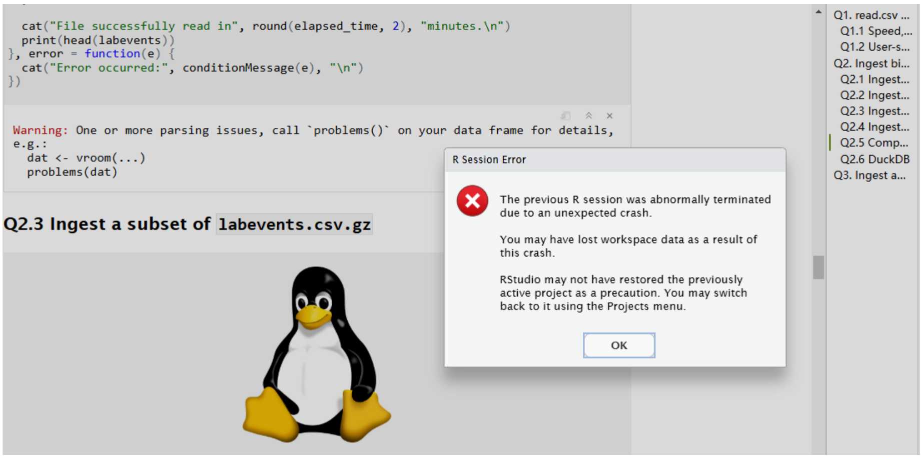
Compressed CSV files (. gz) are read line by line, but will significantly expand in memory. Files like labevents. csv. gz (which may have compressed several GB) may expand to tens of GB during loading. My system doesn't have enough RAM, R will run out of memory and crash. The read_csv() function in the reader reads the entire dataset into memory before processing and ultimately attempts to load all its contents into memory. Unlike databases, it cannot efficiently stream data, resulting in high memory usage and ultimately causing computer memory crashes.

0.2.2 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:

```r
start_time <- Sys.time()

tryCatch({
  labevents <- read_csv("~/mimic/hosp/labevents.csv.gz",
                        col_select = c(subject_id, itemid,
                                       charttime, valuenum))

  end_time <- Sys.time()
  elapsed_time <- difftime(end_time, start_time, units = "mins")

  if (elapsed_time > 3) {
    stop("Reading the file took more than 3 minutes. Aborting process.")
  }

  cat("File successfully read in", round(elapsed_time, 2), "minutes.\n")
  print(head(labevents))
}, error = function(e) {
  cat("Error occurred:", conditionMessage(e), "\n")
})
```



```r
  cat("File successfully read in", round(elapsed_time, 2), "minutes.\n")
  print(head(labevents))
}, error = function(e) {
  cat("Error occurred:", conditionMessage(e), "\n")
})
```

```
Warning: One or more parsing issues, call `problems()` on your data frame for details,
e.g.:
  dat <- vroom(...)
  problems(dat)
```

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

R Session Error

The previous R session was abnormally terminated due to an unexpected crash.

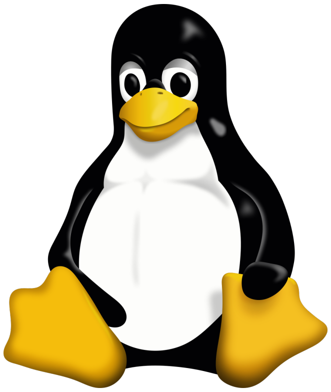
You may have lost workspace data as a result of this crash.

RStudio may not have restored the previously active project as a precaution. You may switch back to it using the Projects menu.

NO, it is the same result with Q2.1.

Even if fewer columns are read, read_csv() will still load all content into RAM. If the dataset has billions of rows and the memory usage is still high, it can cause R to crash. Even if I only selected a few columns, I still have to decompress the entire file first. The labevents.csv.gz file has been compressed, and read_csv() must first decompress it into memory, resulting in high RAM usage. At the same time, unlike databases, read_csv() loads all data into memory at once. It does not support streaming, so even selecting fewer columns cannot fully alleviate memory pressure.

0.2.3 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.)

Solution:

```
zcat < ~/mimic/hosp/labevents.csv.gz | awk -F',' '
    BEGIN { OFS=","; print "subject_id,itemid,charttime,valuenum" }
    NR == 1 {
        for (i=1; i<=NF; i++) { col[$i] = i }
        next
    }
    $col["itemid"] ~ /50912|50971|50983|50902|50882|51221|51301|50931/ {
        print $col["subject_id"],
        $col["itemid"], $col["charttime"], $col["valuenum"]
    }
```

```
' | gzip > 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`?

Solution:

```
# Display the first 10 rows of the file after skipping the header
zcat labevents_filtered.csv.gz | tail -n +2 | head -10

# Count the total number of rows in the file (excluding the header)
expr $(zcat labevents_filtered.csv.gz | wc -l) - 1
```

```
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
10000032,51301,2180-05-06 22:25:00,5
32679896
```

```
# Start measuring time to evaluate performance
start_time <- Sys.time()

# Read the compressed CSV file using readr's read_csv()
# This loads the dataset into memory, expanding the compressed file
labevents_filtered <- read_csv("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.
```

```
# Stop measuring time
end_time <- Sys.time()

# Calculate elapsed time (time taken to read the file)
```

```
elapsed_time <- difftime(end_time, start_time, units = "secs")

cat("Time taken to read labevents_filtered.csv.gz:",
      round(elapsed_time, 2), "seconds\n")
```

```
Time taken to read labevents_filtered.csv.gz: 13.93 seconds
```

```
# Free up memory by removing the tibble
rm(labevents_filtered)
rm(start_time,end_time,elapsed_time)
```

0.2.4 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.

Solution:

```
gzip -d -c ~/mimic/hosp/labevents.csv.gz > \
~/biostat-203b-2025-winter/hw2/labevents.csv
```

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:

```
# 1. Open the CSV dataset efficiently using Apache Arrow
labevents_ds <- open_dataset("~/biostat-203b-2025-winter/hw2/labevents.csv",
                             format = "csv")
```

```
start_time <- Sys.time()

# 2. Perform selection and filtering using dplyr functions
labevents_filtered_arrow <- labevents_ds %>%
  # Select only necessary columns
  select(subject_id, itemid, charttime, valuenum) %>%
  # Filter specific item IDs
  filter(itemid %in% c(50912, 50971, 50983, 50902,
                       50882, 51221, 51301, 50931)) %>%
  collect()#Bring the filtered results into memory as a tibble

end_time <- Sys.time()

elapsed_time <- difftime(end_time, start_time, units = "secs")

cat("Time taken for ingestion, selection, and filtering:",
    round(elapsed_time, 2), "seconds\n")
```

```
Time taken for ingestion, selection, and filtering: 35.75 seconds
```

```
cat("Number of rows in filtered dataset:",
    nrow(labevents_filtered_arrow), "\n")
```

```
Number of rows in filtered dataset: 32679896
```

```
print(labevents_filtered_arrow %>%
  arrange(subject_id, charttime) %>%
    head(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
```

```r
# Free up memory by removing the tibble
rm(labevents_ds,labevents_filtered_arrow)
rm(start_time,end_time,elapsed_time)
```

Apache Arrow is like a high-speed data train. The traditional way of processing large datasets is like moving boxes one after another, while Arrow loads the entire train carriage at once, making the processing speed very fast. It saves data in a format that can be efficiently read by multiple tools (R, Python, Spark) without the need for additional conversions. This means that you can quickly analyze large datasets without running out of memory. If you handle big data, Arrow is your secret weapon for improving speed and efficiency.

0.2.5 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:

```r
# 1. Open the CSV dataset using Apache Arrow
labevents_csv <- open_dataset("~/biostat-203b-2025-winter/hw2/labevents.csv",
                              format = "csv")

# 2. Write the dataset to Parquet format
write_dataset(
  labevents_csv,
  path = "~/biostat-203b-2025-winter/hw2/labevents_parquet",
  format = "parquet"
)
```

```r
# Use the system command to check the size of the Parquet dataset
parquet_size <- system(
  "du -sh ~/biostat-203b-2025-winter/hw2/labevents_parquet/",
  intern = TRUE)
cat("Size of Parquet dataset:", parquet_size, "\n")
```

```
Size of Parquet dataset: 2.6G   /home/gewenqiang/biostat-203b-2025-winter/hw2/labevents_par
```

```
rm(parquet_size)
```

```
# 1. Open the Parquet dataset (created previously)
parquet_data <- open_dataset(
  "~/biostat-203b-2025-winter/hw2/labevents_parquet",
  format = "parquet"
)
```

```
start_time <- Sys.time()
# 2. Time the ingest+select+filter process
result <- parquet_data %>%
  # Select only necessary columns
  select(subject_id, itemid, charttime, valuenum) %>%
  # Filter specific item IDs
  filter(itemid %in% c(50912, 50971, 50983, 50902,
                       50882, 51221, 51301, 50931)) %>%
  collect()#Bring the filtered results into memory as a tibble

end_time <- Sys.time()
elapsed_time <- difftime(end_time, start_time, units = "secs")

cat("Time taken for ingestion, selection, and filtering:",
    round(elapsed_time, 2), "seconds\n")
```

```
Time taken for ingestion, selection, and filtering: 6.13 seconds
```

```
rm(start_time,end_time,elapsed_time)

# 3. Display the number of rows and the first 10 rows
cat("Number of rows in result:", nrow(result), "\n")
```

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

```
print(result %>%
  arrange(subject_id, charttime) %>%
    head(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
```

```
# Free up memory by removing the tibble
rm(result)
```

Parquet makes storing data faster and more efficient, especially for large datasets. Imagine you have a huge spreadsheet, but you only need a few columns instead of the entire file. Parquet does not read data line by line like traditional files such as CSV, but organizes data by column. This means that if you only need one column, it can quickly grab that part without loading everything else - it's a bit like searching for a book in a well-organized library instead of searching through a pile of messy files. It saves storage space and makes data analysis faster! Arrow is primarily an efficient in memory data format designed to accelerate tasks such as data analysis, machine learning, and database queries. Arrow enables data to be stored in RAM in an efficient format, avoiding repeated format conversions and improving query speed. Different languages can directly share data without conversion, with zero copy, greatly improving computational efficiency. Support for vectorized computing: Arrow enables CPU to process data in parallel, faster than traditional methods.

0.2.6 Q2.6 DuckDB

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.

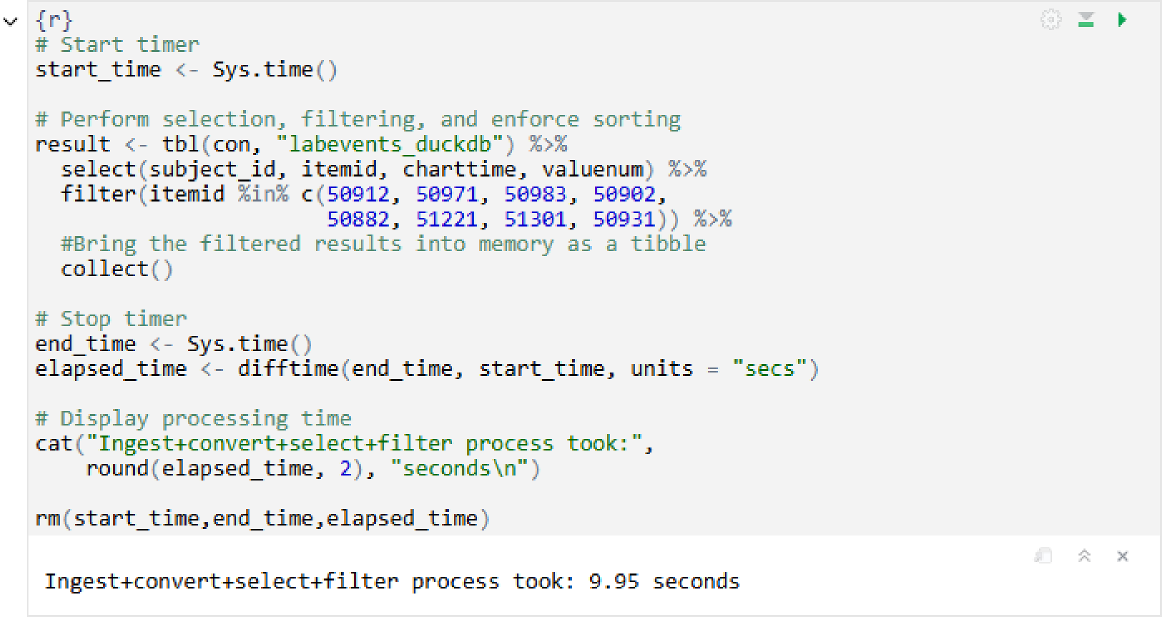
Solution:

```r
# Convert to a DuckDB table
con <- dbConnect(duckdb::duckdb(), dbdir = ":memory:")
duckdb_table <- arrow::to_duckdb(parquet_data,
                                 con = con,
                                 table_name = "labevents_duckdb")
```

```r
# Start timer
start_time <- Sys.time()

# Perform selection, filtering, and enforce sorting
result <- tbl(con, "labevents_duckdb") %>%
  select(subject_id, itemid, charttime, valuenum) %>%
  filter(itemid %in% c(50912, 50971, 50983, 50902,
                       50882, 51221, 51301, 50931)) %>%
  #Bring the filtered results into memory as a tibble
  collect()

# Stop timer
end_time <- Sys.time()
elapsed_time <- difftime(end_time, start_time, units = "secs")

# Display processing time
cat("Ingest+convert+select+filter process took:",
    round(elapsed_time, 2), "seconds\n")

rm(start_time,end_time,elapsed_time)
```

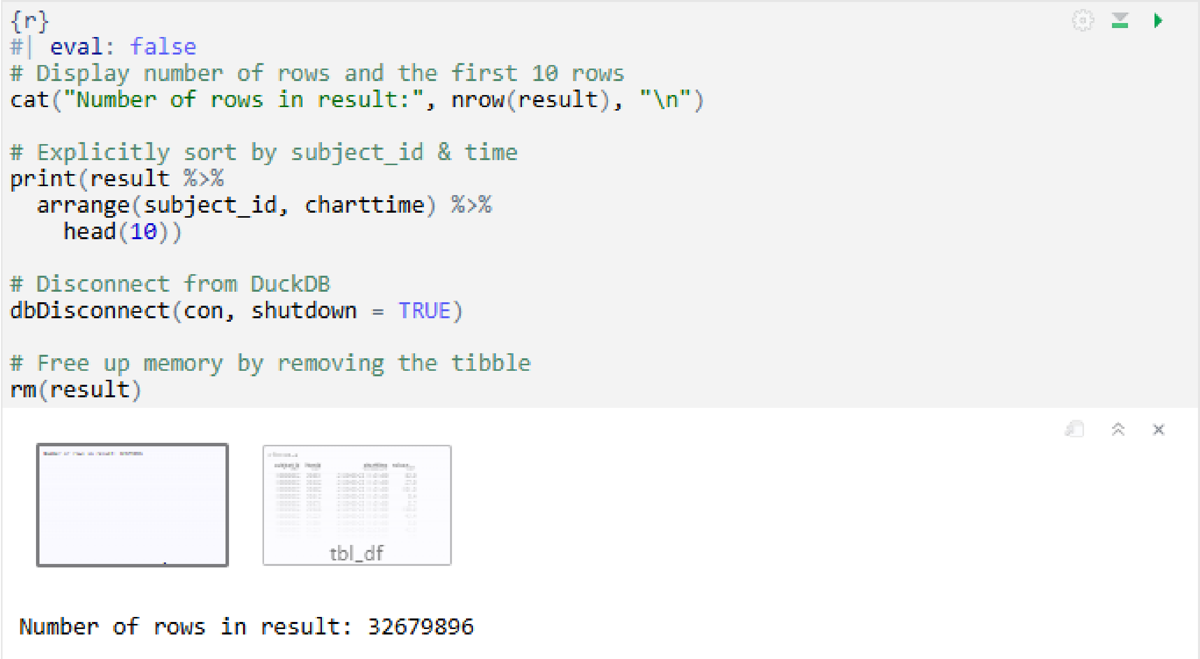
```r
# Display number of rows and the first 10 rows
cat("Number of rows in result:", nrow(result), "\n")

# Explicitly sort by subject_id & time
print(result %>%
  arrange(subject_id, charttime) %>%
    head(10))

# Disconnect from DuckDB
dbDisconnect(con, shutdown = TRUE)

# Free up memory by removing the tibble
rm(result)
```

Chunks' results:



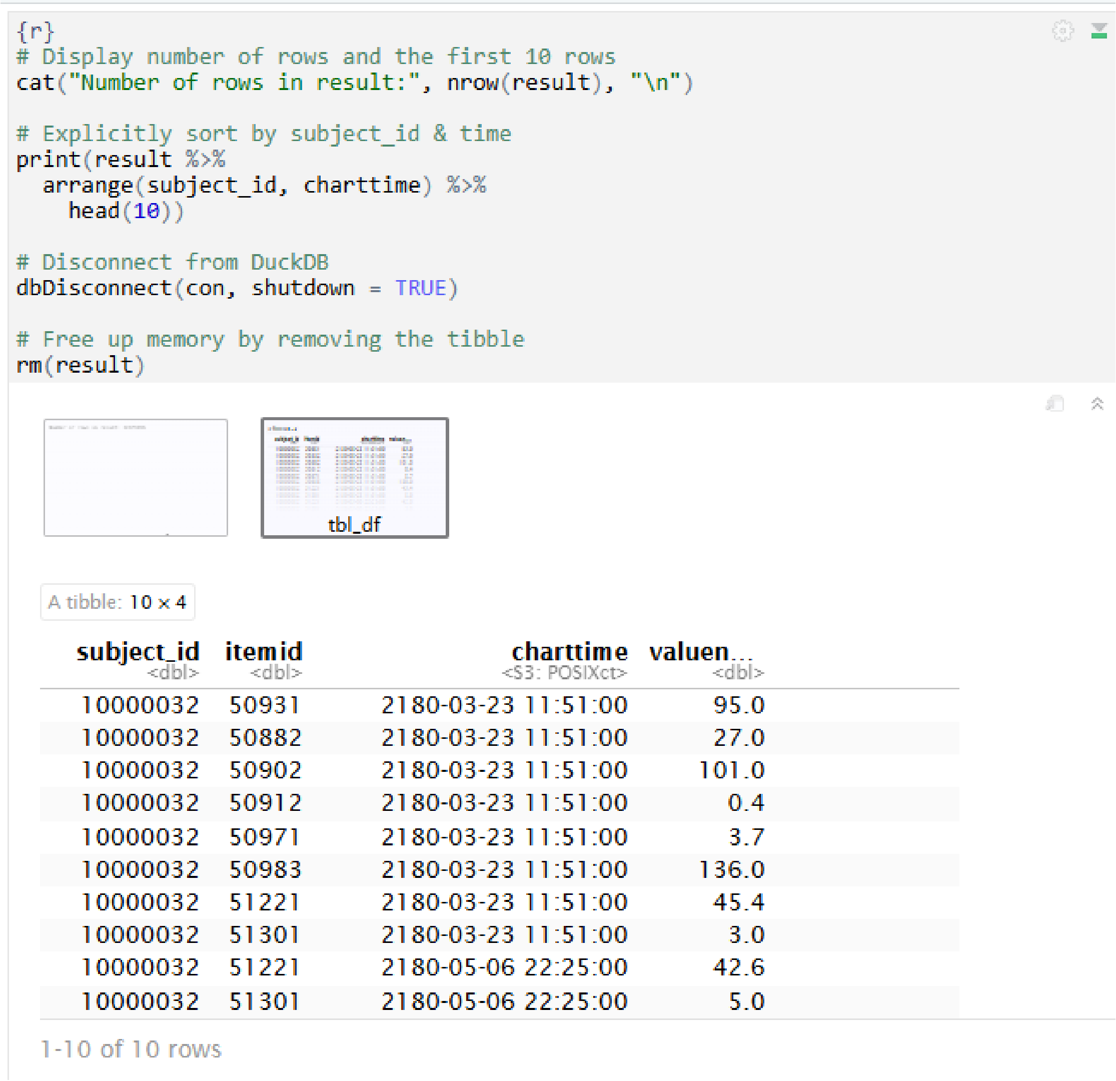
```
{r}
# Start timer
start_time <- Sys.time()

# Perform selection, filtering, and enforce sorting
result <- tbl(con, "labevents_duckdb") %>%
  select(subject_id, itemid, charttime, valuenum) %>%
  filter(itemid %in% c(50912, 50971, 50983, 50902,
                       50882, 51221, 51301, 50931)) %>%
  #Bring the filtered results into memory as a tibble
  collect()

# Stop timer
end_time <- Sys.time()
elapsed_time <- difftime(end_time, start_time, units = "secs")

# Display processing time
cat("Ingest+convert+select+filter process took:",
    round(elapsed_time, 2), "seconds\n")

rm(start_time,end_time,elapsed_time)
```

```
Ingest+convert+select+filter process took: 9.95 seconds
```



```
{r}
#| eval: false
# Display number of rows and the first 10 rows
cat("Number of rows in result:", nrow(result), "\n")

# Explicitly sort by subject_id & time
print(result %>%
  arrange(subject_id, charttime) %>%
    head(10))

# Disconnect from DuckDB
dbDisconnect(con, shutdown = TRUE)

# Free up memory by removing the tibble
rm(result)
```

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



```
{r}
# Display number of rows and the first 10 rows
cat("Number of rows in result:", nrow(result), "\n")

# Explicitly sort by subject_id & time
print(result %>%
  arrange(subject_id, charttime) %>%
    head(10))

# Disconnect from DuckDB
dbDisconnect(con, shutdown = TRUE)

# Free up memory by removing the tibble
rm(result)
```

A tibble: 10 x 4

subject_id <dbl>	itemid <dbl>	charttime <S3: POSIXct>	valuen... <dbl>
10000032	50931	2180-03-23 11:51:00	95.0
10000032	50882	2180-03-23 11:51:00	27.0
10000032	50902	2180-03-23 11:51:00	101.0
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	136.0
10000032	51221	2180-03-23 11:51:00	45.4
10000032	51301	2180-03-23 11:51:00	3.0
10000032	51221	2180-05-06 22:25:00	42.6
10000032	51301	2180-05-06 22:25:00	5.0

1-10 of 10 rows

Imagine you have a bunch of files, and every time you need to find something, you have to flip through them one by one - this is how traditional databases work. On the other hand** DuckDB * * is like having a super fast librarian who will immediately give you what you need. It is designed for lightning fast data analysis, especially for large datasets, without the hassle of setting up a large database server. You can run it directly on your laptop, just like opening a spreadsheet, but its working speed is comparable to modern data tools. It's like giving your data a * * turbocharger * *, making complex queries feel as simple as Google search!

0.3 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
```

How many rows? 433 millions.(432997491)

```
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
```

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: Using the Apache Arrow

```
# Define file paths
chartevents_path <- "~/mimic/icu/chartevents.csv.gz"
parquet_path <- "~/biostat-203b-2025-winter/hw2/chartevents_vitals.parquet"
```

```
# Start timing
start_time <- Sys.time()

# Load and filter using Apache Arrow
chartevents_vitals <- open_dataset(chartevents_path, format = "csv") %>%
  select(subject_id, itemid, charttime,valuenum) %>%
  filter(itemid %in% c(220045, 220181, 220179, 223761, 220210)) %>%
  collect()  # Convert to in-memory tibble

# End timing
end_time <- Sys.time()
elapsed_time <- difftime(end_time, start_time, units = "secs")

# Display processing time
cat("Time taken for filtering and saving:",
    round(elapsed_time, 2), "seconds\n")
```

```
Time taken for filtering and saving: 105.68 seconds
```

```
rm(start_time,end_time,elapsed_time)
```

```
# Display number of rows in the filtered dataset
cat("Number of rows in filtered dataset:",
    nrow(chartevents_vitals), "\n")
```

```
Number of rows in filtered dataset: 30195426
```

```
# Display first 10 rows
print(head(chartevents_vitals, 10))
```

```
# A tibble: 10 x 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
```

```
# Free up memory by removing the tibble
rm(chartevents_vitals)
```