Biostat 203B Homework 2

Due Feb 7, 2025 @ 11:59PM

AUTHOR
Sophia Luo, 106409469

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_1.0.1
```

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

```
library(arrow)
```

```

Attaching package: 'arrow'

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

    timestamp
```

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

```
Loading required package: DBI
```

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

```
Attaching package: 'pryr'

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

    address
```

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

```
Loading required package: R.oo

Loading required package: R.methodsS3

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

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


Attaching package: 'R.oo'

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

    throw

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

    getClasses, getMethods

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

    attach, detach, load, save

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


Attaching package: 'R.utils'

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

    timestamp
```

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

    timestamp

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

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

```r
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

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

```
Totalram:  7.633 GiB
Freeram:   5.129 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 mmmm2627 mmmm2627   19928140 Jan 16 12:39 admissions.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627     427554 Jan 16 12:39 d_hcpcs.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627     876360 Jan 16 12:39 d_icd_diagnoses.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627     589186 Jan 16 12:39 d_icd_procedures.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627      13169 Jan 16 12:39 d_labitems.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627   33564802 Jan 16 12:39 diagnoses_icd.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627    9743908 Jan 16 12:39 drgcodes.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627  811305629 Jan 16 12:39 emar.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627  748158322 Jan 16 12:39 emar_detail.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627    2162335 Jan 16 12:39 hcpcsevents.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 2592909134 Jan 16 12:39 labevents.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627  174144176 Jan 30 15:28 labevents_filtered.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627  117644075 Jan 16 12:39 microbiologyevents.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627   44069351 Jan 16 12:39 omr.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627    2835586 Jan 16 12:39 patients.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627  525708076 Jan 16 12:39 pharmacy.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627  666594177 Jan 16 12:39 poe.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627   55267894 Jan 16 12:39 poe_detail.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627  606298611 Jan 16 12:39 prescriptions.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627    7777324 Jan 16 12:39 procedures_icd.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627     127330 Jan 16 12:39 provider.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627    8569241 Jan 16 12:39 services.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627   46185771 Jan 16 12:39 transfers.csv.gz
```

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

```
total 4253392
-rwxrwxrwx 1 mmmm2627 mmmm2627      41566 Jan 16 12:39 caregiver.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 3502392765 Jan 16 12:40 chartevents.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627      58741 Jan 16 12:40 d_items.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627   63481196 Jan 16 12:40 datetimeevents.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627    3342355 Jan 16 12:40 icustays.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627  311642048 Jan 16 12:40 ingredientevents.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627  401088206 Jan 16 12:40 inputevents.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627   49307639 Jan 16 12:40 outputevents.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627   24096834 Jan 16 12:40 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:

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

time_base <- system.time(df_base <- read.csv(file_path))
size_base <- object_size(df_base)

time_tidy <- system.time(df_tidy <- read_csv(file_path, show_col_types = FALSE))
size_tidy <- object_size(df_tidy)

time_dt <- system.time(df_dt <- fread(file_path))
size_dt <- object_size(df_dt)
```

```
results <- data.frame(
  Function = c("read.csv", "read_csv", "fread"),
  Time = c(time_base[3], time_tidy[3], time_dt[3]),
  Memory_Usage = c(size_base, size_tidy, size_dt)
)
print(results)
```

```
  Function   Time Memory_Usage
1 read.csv 10.443    200.10 MB
2 read_csv  1.932     70.02 MB
3    fread  1.363     63.47 MB
```

`fread` appears to be the fastest function and takes least amount of memory. `read.csv` is the slowest function and takes most amount of memory.

```
print("Data type parsed by base R:")
```

```
[1] "Data type parsed by base R:"
```

```
str(df_base)
```

```
'data.frame':	546028 obs. of  16 variables:
 $ subject_id          : int  10000032 10000032 10000032 10000032 10000068 10000084 10000084
10000108 10000117 10000117 ...
 $ hadm_id             : int  22595853 22841357 25742920 29079034 25022803 23052089 29888819
27250926 22927623 27988844 ...
 $ admittime           : chr  "2180-05-06 22:23:00" "2180-06-26 18:27:00" "2180-08-05 23:44:00"
```

```
"2180-07-23 12:35:00" ...
 $ dischtime           : chr  "2180-05-07 17:15:00" "2180-06-27 18:49:00" "2180-08-07 17:50:00"
"2180-07-25 17:55:00" ...
 $ deathtime           : chr  "" "" "" "" ...
 $ admission_type      : chr  "URGENT" "EW EMER." "EW EMER." "EW EMER." ...
 $ admit_provider_id   : chr  "P49AFC" "P784FA" "P19UTS" "P06OTX" ...
 $ admission_location  : chr  "TRANSFER FROM HOSPITAL" "EMERGENCY ROOM" "EMERGENCY ROOM"
"EMERGENCY ROOM" ...
 $ discharge_location  : chr  "HOME" "HOME" "HOSPICE" "HOME" ...
 $ insurance           : chr  "Medicaid" "Medicaid" "Medicaid" "Medicaid" ...
 $ language            : chr  "English" "English" "English" "English" ...
 $ marital_status      : chr  "WIDOWED" "WIDOWED" "WIDOWED" "WIDOWED" ...
 $ race                : chr  "WHITE" "WHITE" "WHITE" "WHITE" ...
 $ edregtime           : chr  "2180-05-06 19:17:00" "2180-06-26 15:54:00" "2180-08-05 20:58:00"
"2180-07-23 05:54:00" ...
 $ edouttime           : chr  "2180-05-06 23:30:00" "2180-06-26 21:31:00" "2180-08-06 01:44:00"
"2180-07-23 14:00:00" ...
 $ hospital_expire_flag: int  0 0 0 0 0 0 0 0 0 0 ...
```

```
print("Data type parsed by tidyverse:")
```

```
[1] "Data type parsed by tidyverse:"
```

```
str(df_tidy)
```

```
spc_tbl_ [546,028 × 16] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ subject_id          : num [1:546028] 1e+07 1e+07 1e+07 1e+07 1e+07 ...
 $ hadm_id             : num [1:546028] 22595853 22841357 25742920 29079034 25022803 ...
 $ admittime           : POSIXct[1:546028], format: "2180-05-06 22:23:00" "2180-06-26 18:27:00"
...
 $ dischtime           : POSIXct[1:546028], format: "2180-05-07 17:15:00" "2180-06-27 18:49:00"
...
 $ deathtime           : POSIXct[1:546028], format: NA NA ...
 $ admission_type      : chr [1:546028] "URGENT" "EW EMER." "EW EMER." "EW EMER." ...
 $ admit_provider_id   : chr [1:546028] "P49AFC" "P784FA" "P19UTS" "P06OTX" ...
 $ admission_location  : chr [1:546028] "TRANSFER FROM HOSPITAL" "EMERGENCY ROOM" "EMERGENCY
ROOM" "EMERGENCY ROOM" ...
 $ discharge_location  : chr [1:546028] "HOME" "HOME" "HOSPICE" "HOME" ...
 $ insurance           : chr [1:546028] "Medicaid" "Medicaid" "Medicaid" "Medicaid" ...
 $ language            : chr [1:546028] "English" "English" "English" "English" ...
 $ marital_status      : chr [1:546028] "WIDOWED" "WIDOWED" "WIDOWED" "WIDOWED" ...
 $ race                : chr [1:546028] "WHITE" "WHITE" "WHITE" "WHITE" ...
 $ edregtime           : POSIXct[1:546028], format: "2180-05-06 19:17:00" "2180-06-26 15:54:00"
...
 $ edouttime           : POSIXct[1:546028], format: "2180-05-06 23:30:00" "2180-06-26 21:31:00"
...
 $ hospital_expire_flag: num [1:546028] 0 0 0 0 0 0 0 0 0 0 ...
 - attr(*, "spec")=
  .. cols(
```

```
  ..   subject_id = col_double(),
  ..   hadm_id = col_double(),
  ..   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_double()
  .. )
 - attr(*, "problems")=<externalptr>
```

```
print("Data type parsed by data.table:")
```

```
[1] "Data type parsed by data.table:"
```

```
str(df_dt)
```

```
Classes 'data.table' and 'data.frame':  546028 obs. of  16 variables:
 $ subject_id          : int  10000032 10000032 10000032 10000032 10000068 10000084 10000084
10000108 10000117 10000117 ...
 $ hadm_id             : int  22595853 22841357 25742920 29079034 25022803 23052089 29888819
27250926 22927623 27988844 ...
 $ admittime           : POSIXct, format: "2180-05-06 22:23:00" "2180-06-26 18:27:00" ...
 $ dischtime           : POSIXct, format: "2180-05-07 17:15:00" "2180-06-27 18:49:00" ...
 $ deathtime           : POSIXct, format: NA NA ...
 $ admission_type      : chr  "URGENT" "EW EMER." "EW EMER." "EW EMER." ...
 $ admit_provider_id   : chr  "P49AFC" "P784FA" "P19UTS" "P06OTX" ...
 $ admission_location  : chr  "TRANSFER FROM HOSPITAL" "EMERGENCY ROOM" "EMERGENCY ROOM"
"EMERGENCY ROOM" ...
 $ discharge_location  : chr  "HOME" "HOME" "HOSPICE" "HOME" ...
 $ insurance           : chr  "Medicaid" "Medicaid" "Medicaid" "Medicaid" ...
 $ language            : chr  "English" "English" "English" "English" ...
 $ marital_status      : chr  "WIDOWED" "WIDOWED" "WIDOWED" "WIDOWED" ...
 $ race                : chr  "WHITE" "WHITE" "WHITE" "WHITE" ...
 $ edregtime           : POSIXct, format: "2180-05-06 19:17:00" "2180-06-26 15:54:00" ...
 $ edouttime           : POSIXct, format: "2180-05-06 23:30:00" "2180-06-26 21:31:00" ...
 $ hospital_expire_flag: int  0 0 0 0 0 0 0 0 0 0 ...
 - attr(*, ".internal.selfref")=<externalptr>
```

`read.csv` parses data to be a `data.frame` with either `int` or `chr` data type. `read_csv` and `fread` parses data to be a `data.frame` with `int` or `chr` data type. Interestingly, both `read_csv` and `fread` recognize data that are date and set it to `POSIXct` data type. Additionally, `tidyverse` parses `subject_id` mistakenly to number in scientific notation.

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:

```
col_types <- cols(
  subject_id = col_character(),
  hadm_id = col_character(),
  admittime = col_datetime(),
  dischtime = col_datetime(),
  deathtime = col_datetime(),
  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(),
  edouttime = col_datetime(),
  hospital_expire_flag = col_integer()
)

time_specified <- system.time(
  df_specified <- read_csv(file_path, col_types = col_types)
)
size_specified <- object_size(df_specified)

time_specified
```

```
   user  system elapsed 
  1.697   0.260   3.627 
```

```
size_specified
```

```
117.09 MB
```

Both running time and memory usage increases when column types are specified. The run time increases from 1.653 to 2.060 seconds. It takes 117.09 MB memory to read the data.

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 mmmm2627 mmmm2627 2592909134 Jan 16 12:39 /home/mmmm2627/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.

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

Note: `eval=FALSE` is set to avoid program crashing during rendering.

```
system.time(labevents <- read_csv(file_path))
```

My RStudio program crashed before reaching 3 minutes. This is because the file size is so big that it exceeds the memory of my laptop to process it.

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`.)

Note: `eval=FALSE` is set to avoid program crashing during rendering.

```
read_csv(file_path, col_select=c("subject_id","itemid", "charttime","valuenum"))
```

My RStudio program crashed again. Even after selecting specific columns, the program still needs to process large size file and it crashes after exceeding maximum memory of my laptop.

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 |
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 of the new file:

```
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 in the new file, excluding the header:

Note: Caching is used here to avoid long running and memory overload issue during rendering.

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

```
32679896
```

Time for `read_csv` to ingest filtered file:

Note: Caching is used here to avoid long running and memory overload issue during rendering.

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

system.time(labevents <- read_csv(file_path, show_col_types = FALSE))
```

```
   user  system elapsed
 73.007  32.480  41.637
```

It took about 25 seconds for `read_csv` to ingest the filtered file.

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:

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

Note: Caching is used here to avoid long running and memory overload issue during rendering.

```r
system.time({
  labevents <- open_dataset("labevents.csv", format = "csv")

  labevents_arrow <- labevents |>
    select(subject_id, itemid, charttime, valuenum) |>
    filter(itemid %in% c(50912, 50971, 50983, 50902, 50882, 51221, 51301, 50931)) |>
    collect()
})
```

```
   user  system elapsed 
 45.714  10.578  39.975 
```

It takes 64 seconds to ingest, select, and filter content in `labevents.csv`.

Display the number of rows:

```r
nrow(labevents_arrow)
```

```
[1] 32679896
```

Display first 10 rows of the result tibble:

```r
head(labevents_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  
```

The number of lines and the first 10 rows of the result tibble matches those in Q2.3

Note: `labevents_arrow` is removed after printing the first 10 rows to save memory and avoid out of memory issue during rendering.

```r
rm(labevents_arrow)
gc() # Force garbage collection
```

```
          used  (Mb) gc trigger   (Mb)  max used   (Mb)
Ncells  4058463 216.8    7879948  420.9   7879948  420.9
```

```
Vcells 48631072 371.1  246182660 1878.3 287375911 2192.6
```

Apache Arrow is a lightning-fast data processing framework that allows efficient handling of large datasets without loading everything into memory. It does this by using a columnar in-memory format, which makes operations like filtering and selecting data extremely fast. Think of it as a highway for data—allowing seamless, high-speed movement between different tools like R, Python, and databases. Instead of copying data between systems (which slows things down), Arrow lets them share the same memory, making everything much more efficient.

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`](https://arrow.apache.org/docs/r/reference/write_dataset.html).) 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:

Re-write the csv file in the binary Parquet format:

Note: `eval` is set to `FALSE` to avoid long running time in rendering

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

write_dataset(labevents, "labevents_parquet", format = "parquet")
```

```bash
ls -lh labevents_parquet
```

```
total 2.6G
-rw-r--r-- 1 mmmm2627 mmmm2627 2.6G Jan 31 01:00 part-0.parquet
```

The Parquet file is 2.6G.

Ingest, select, and filter Parquet file:

Note: `cache` is used to save rendering time.

```r
system.time({
  labevents_parquet <- open_dataset("labevents_parquet", format = "parquet")

  labevents_filtered_parquet <- labevents_parquet %>%
    select(subject_id, itemid, charttime, valuenum) %>%
    filter(itemid %in% c(50912, 50971, 50983, 50902, 50882, 51221, 51301, 50931)) %>%
    collect()  # Load into memory
})
```

```
   user  system elapsed 
 27.553  18.788  10.293 
```

It took 11 seconds to ingest, select, and filter Parquet file.

Display the number of rows:

```
nrow(labevents_filtered_parquet)
```

```
[1] 32679896
```

First 10 rows of result tibble:

```
head(labevents_filtered_parquet, 10)
```

```
# A tibble: 10 × 4
   subject_id itemid charttime           valuenum
        <int>  <int> <dttm>                 <dbl>
 1   10001884  50971 2130-04-08 11:15:00      3.8
 2   10001884  50983 2130-04-08 11:15:00    138  
 3   10001884  51221 2130-04-08 11:15:00     40.2
 4   10001884  51301 2130-04-08 11:15:00      5.7
 5   10001884  50882 2130-04-08 22:55:00     29  
 6   10001884  50902 2130-04-08 22:55:00     99  
 7   10001884  50912 2130-04-08 22:55:00      0.8
 8   10001884  50931 2130-04-08 22:55:00    149  
 9   10001884  50971 2130-04-08 22:55:00      4.5
10   10001884  50983 2130-04-08 22:55:00    137  
```

This verifies that the number of rows and the first 10 rows matches those in Q2.3.

Parquet is a high-performance, space-efficient file format designed for big data. Unlike traditional CSV, Parquet stores data column-wise instead of row-wise. This makes it much faster for analytics, because when you filter or select specific columns, you don't need to read the entire file—only the relevant parts. Parquet also compresses data better than CSV, saving storage space while boosting performance. Think of it as a well-organized, indexed library, where you can quickly find the books (data) you need instead of scanning every shelf.

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:

Note: `cache` is used to reduce rendering time.

```
system.time({
  # Ingest Parquet dataset
  labevents_parquet <- open_dataset("labevents_parquet", format = "parquet")

  # Convert to DuckDB table
  con <- dbConnect(duckdb::duckdb(), dbdir = ":memory:")
  labevents_duckdb <- to_duckdb(labevents_parquet, con)

  # Select columns and filter rows
  labevents_filtered_duckdb <- labevents_parquet |>
    select(subject_id, itemid, charttime, valuenum) |>
    filter(itemid %in% c(50912, 50971, 50983, 50902, 50882, 51221, 51301, 50931)) %>%
    collect()

  # Close DuckDB connection
  dbDisconnect(con)
})
```

```
   user  system elapsed 
 54.619  43.438  15.682 
```

It took 19 seconds to ingest Parquet file, select columns, and filter rows.

Display the number of rows:

```
nrow(labevents_filtered_duckdb)
```

```
[1] 32679896
```

Display first 10 rows:

```
head(labevents_filtered_parquet, 10)
```

```
# A tibble: 10 × 4
   subject_id itemid charttime           valuenum
        <int>  <int> <dttm>                 <dbl>
 1   10001884  50971 2130-04-08 11:15:00      3.8
 2   10001884  50983 2130-04-08 11:15:00    138  
 3   10001884  51221 2130-04-08 11:15:00     40.2
 4   10001884  51301 2130-04-08 11:15:00      5.7
 5   10001884  50882 2130-04-08 22:55:00     29  
 6   10001884  50902 2130-04-08 22:55:00     99  
 7   10001884  50912 2130-04-08 22:55:00      0.8
 8   10001884  50931 2130-04-08 22:55:00    149  
 9   10001884  50971 2130-04-08 22:55:00      4.5
10   10001884  50983 2130-04-08 22:55:00    137  
```

This confirms that DuckDB generated file matches those in Q2.3.

Note: `labevents_filtered_parquet` is removed after printing the first 10 rows to save memory and avoid out of memory issue during rendering.

```
rm(labevents_filtered_parquet)
gc()
```

```
            used   (Mb) gc trigger   (Mb)  max used   (Mb)
Ncells   4421826  236.2    7879948  420.9   7879948  420.9
Vcells 147331357 1124.1  425564967 3246.9 517057875 3944.9
```

DuckDB is a fast, lightweight database designed for efficient data analysis on a single machine. Unlike traditional databases that optimize for many users, DuckDB is built for analytics—it processes large datasets blazingly fast using an optimized columnar format. It's like having the power of a full-fledged database engine without needing a server. Imagine Excel on steroids, where queries run instantly, and we can work with billions of rows seamlessly.

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` to `chartevents.csv` into current directory

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

Compress `chartevents.csv` to Parquet format:

Note: `eval` is set to `FALSE` to reduce long rendering time and avoid memery overload.

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

write_dataset(chartevents, path = "chartevents_parquet", format = "parquet")
```

Convert Parquet to DuckDB & filter data:

```
chartevents_parquet <- open_dataset("chartevents_parquet", format = "parquet")

con <- dbConnect(duckdb::duckdb(), dbdir = ":memory:")
chartevents_duckdb <- to_duckdb(chartevents_parquet, con)

chartevents_filtered_duckdb <- chartevents_duckdb |>
  select(subject_id, itemid, charttime, valuenum) |>
  filter(itemid %in% c(220045,220181,220179,223761,220210)) |>
  collect()

dbDisconnect(con)
```

Display the number of rows

```
nrow(chartevents_filtered_duckdb)
```

```
[1] 30195426
```

There are 30195426 rows in the filtered file.

Display the first 10 rows:

```
head(chartevents_filtered_duckdb, 10)
```

```
# A tibble: 10 × 4
   subject_id itemid charttime           valuenum
        <dbl>  <dbl> <dttm>                 <dbl>
 1   10003400 220045 2137-08-13 09:00:00      104
 2   10003400 220179 2137-08-13 09:00:00       94
 3   10003400 220181 2137-08-13 09:00:00       64
 4   10003400 220210 2137-08-13 09:00:00       22
 5   10003400 220045 2137-08-13 10:00:00       91
 6   10003400 220179 2137-08-13 10:00:00       91
 7   10003400 220181 2137-08-13 10:00:00       66
 8   10003400 220210 2137-08-13 10:00:00       18
 9   10003400 220045 2137-08-13 11:00:00       84
10   10003400 220179 2137-08-13 11:00:00       93
```

Personal thoughts: Even though using DuckDB to ingest Parquet file and the process data is the fastest method in ingesting and manipulating data, the prerequisite steps can be quite time consuming. 1) Decompress `.gz` file 2) Re-write `.csv` file in Parquet format. These two steps are time costly. If we will do lots of data manipulation later, then these steps are worth it. Otherwise, simply use `read_csv` might be a better way for one time access.