Biostat 203B Homework 2

Due Feb 7, 2025 @ 11:59PM

Emma Mo and 906542365

Display machine information for reproducibility:

```
sessionInfo()
```

```
R version 4.4.1 (2024-06-14)
Platform: x86_64-apple-darwin20
Running under: macOS Monterey 12.4

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

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

time zone: Asia/Shanghai
tzcode source: internal

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

loaded via a namespace (and not attached):
 [1] compiler_4.4.1    fastmap_1.2.0     cli_3.6.3         tools_4.4.1
 [5] htmltools_0.5.8.1 rstudioapi_0.16.0 yaml_2.3.10       rmarkdown_2.29
 [9] knitr_1.49        jsonlite_1.8.9    xfun_0.50         digest_0.6.37
[13] rlang_1.1.4       evaluate_1.0.3
```

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

```
library(arrow)
```

```
Attaching package: 'arrow'

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

    timestamp
```

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

```
Loading required package: DBI
```

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

```
Attaching package: 'pryr'

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

    address
```

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

```
Loading required package: R.oo

Loading required package: R.methodsS3

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

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

Attaching package: 'R.oo'
```

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

    throw

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

    getClasses, getMethods

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

    attach, detach, load, save

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


Attaching package: 'R.utils'

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

    timestamp

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

    timestamp

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

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

```
library(tidyverse)
```

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

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

Display memory information of your computer

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

```
Totalram:  16.000 GiB
Freeram:    3.890 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 12306760
-rw-r--r--  1 emma  staff    19928140 Jun 25  2024 admissions.csv.gz
```

```
-rw-r--r--  1 emma  staff      427554 Apr 13  2024 d_hcpcs.csv.gz
-rw-r--r--  1 emma  staff      876360 Apr 13  2024 d_icd_diagnoses.csv.gz
-rw-r--r--  1 emma  staff      589186 Apr 13  2024 d_icd_procedures.csv.gz
-rw-r--r--  1 emma  staff       13169 Oct  4 00:07 d_labitems.csv.gz
-rw-r--r--  1 emma  staff    33564802 Oct  4 00:07 diagnoses_icd.csv.gz
-rw-r--r--  1 emma  staff     9743908 Oct  4 00:07 drgcodes.csv.gz
-rw-r--r--  1 emma  staff   811305629 Apr 13  2024 emar.csv.gz
-rw-r--r--  1 emma  staff   748158322 Apr 13  2024 emar_detail.csv.gz
-rw-r--r--  1 emma  staff     2162335 Apr 13  2024 hcpcsevents.csv.gz
-rw-r--r--  1 emma  staff  2592909134 Oct  4 00:08 labevents.csv.gz
-rw-r--r--  1 emma  staff      262144 Jan 31 17:05 labevents_filtered.csv.gz
-rw-r--r--  1 emma  staff   117644075 Oct  4 00:08 microbiologyevents.csv.gz
-rw-r--r--  1 emma  staff    44069351 Oct  4 00:08 omr.csv.gz
-rw-r--r--  1 emma  staff     2835586 Apr 13  2024 patients.csv.gz
-rw-r--r--  1 emma  staff   525708076 Apr 13  2024 pharmacy.csv.gz
-rw-r--r--  1 emma  staff   666594177 Apr 13  2024 poe.csv.gz
-rw-r--r--  1 emma  staff    55267894 Apr 13  2024 poe_detail.csv.gz
-rw-r--r--  1 emma  staff   606298611 Apr 13  2024 prescriptions.csv.gz
-rw-r--r--  1 emma  staff     7777324 Apr 13  2024 procedures_icd.csv.gz
-rw-r--r--  1 emma  staff      127330 Apr 13  2024 provider.csv.gz
-rw-r--r--  1 emma  staff     8569241 Apr 13  2024 services.csv.gz
-rw-r--r--  1 emma  staff    46185771 Oct  4 00:08 transfers.csv.gz
```

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

```
total 8506784
-rw-r--r--  1 emma  staff       41566 Apr 13  2024 caregiver.csv.gz
-rw-r--r--  1 emma  staff  3502392765 Apr 13  2024 chartevents.csv.gz
drwxr-xr-x  3 emma  staff          96 Feb  1 12:31 csv_file
-rw-r--r--  1 emma  staff       58741 Apr 13  2024 d_items.csv.gz
-rw-r--r--  1 emma  staff    63481196 Apr 13  2024 datetimeevents.csv.gz
-rw-r--r--  1 emma  staff     3342355 Oct  3 22:36 icustays.csv.gz
-rw-r--r--  1 emma  staff   311642048 Apr 13  2024 ingredientevents.csv.gz
-rw-r--r--  1 emma  staff   401088206 Apr 13  2024 inputevents.csv.gz
-rw-r--r--  1 emma  staff    49307639 Apr 13  2024 outputevents.csv.gz
-rw-r--r--  1 emma  staff    24096834 Apr 13  2024 procedureevents.csv.gz
```

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

Q1.1 Speed, memory, and data types

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

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

library(tidyverse)
library(data.table)
library(pryr)
```

```
# Measure speed and memory usage for read.csv (Base R)
system.time(df_base <- read.csv(admissions))
```

```
   user  system elapsed 
 10.403   0.161  10.599 
```

```
memory_base <- object_size(df_base)
str(df_base)
```

```
'data.frame':   546028 obs. of  16 variables:
 $ subject_id          : int  10000032 10000032 10000032 10000032 10000068 10000084 10000084
 $ hadm_id             : int  22595853 22841357 25742920 29079034 25022803 23052089 29888819
 $ admittime           : chr  "2180-05-06 22:23:00" "2180-06-26 18:27:00" "2180-08-05 23:44:0
 $ dischtime           : chr  "2180-05-07 17:15:00" "2180-06-27 18:49:00" "2180-08-07 17:50:0
 $ 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" "EM
 $ 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 20:58:0
 $ edouttime           : chr  "2180-05-06 23:30:00" "2180-06-26 21:31:00" "2180-08-06 01:44:0
 $ hospital_expire_flag: int  0 0 0 0 0 0 0 0 0 0 ...
```

```
memory_base
```

```
200.10 MB
```

- speed: 5.676s
- memory: 200.1 MB
- data type: int and chr (string)

```
# Measure speed and memory usage for read_csv (tidyverse)
system.time(df_tidy <- read_csv(admissions))
```

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

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

   user  system elapsed
  3.141   1.595   1.457
```

```
memory_tidy <- object_size(df_tidy)
str(df_tidy)
```

```
spc_tbl_ [546,028 x 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 18:27:0
 $ dischtime           : POSIXct[1:546028], format: "2180-05-07 17:15:00" "2180-06-27 18:49:0
 $ 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
 $ 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 15:54:0
 $ edouttime           : POSIXct[1:546028], format: "2180-05-06 23:30:00" "2180-06-26 21:31:0
 $ 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>
```

```
memory_tidy
```

```
70.02 MB
```

- speed: 1.432s
- memory: 70.02 MB
- data type: num, chr (string), POSIXct (datetime)

```
# Measure speed and memory usage for fread (data.table)
system.time(df_dt <- fread(admissions))
```

```
   user  system elapsed 
  1.685   0.343   0.983 
```

```
memory_dt <- object_size(df_dt)
str(df_dt)
```

```
Classes 'data.table' and 'data.frame':  546028 obs. of  16 variables:
 $ subject_id          : int  10000032 10000032 10000032 10000032 10000068 10000084 10000084
 $ hadm_id             : int  22595853 22841357 25742920 29079034 25022803 23052089 29888819
 $ 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" "EM
 $ 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>
```

```
memory_dt
```

```
63.47 MB
```

- speed: 0.979s
- memory: 63.47 MB
- data type: int, chr (string), POSIXct (datetime)

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.)

- `fread` in the data.table package is the fastest and uses the least memory.
- `read.csv` in base R is the slowest and uses the largest memory.
- Base R (read.csv) treats all dates (admittime, dischtime, etc.) as characters (chr), requiring manual conversion to datetime. Tidyverse (read_csv) and data.table (fread) automatically parse them as POSIXct (datetime), making them immediately usable for date calculations.

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`.)

```
col_types_spec <- 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()
)

system.time(df_tidy_optimized <- read_csv(admissions, col_types = col_types_spec))
```

```
   user  system elapsed
  3.034   1.085   1.384
```

```
memory_tidy_optimized <- object_size(df_tidy_optimized)
str(df_tidy_optimized)
```

```
spc_tbl_ [546,028 x 16] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ subject_id          : int [1:546028] 10000032 10000032 10000032 10000032 10000068 1000008
 $ hadm_id             : int [1:546028] 22595853 22841357 25742920 29079034 25022803 2305208
 $ admittime           : POSIXct[1:546028], format: "2180-05-06 22:23:00" "2180-06-26 18:27:
 $ dischtime           : POSIXct[1:546028], format: "2180-05-07 17:15:00" "2180-06-27 18:49:
 $ 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
 $ 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 15:54:0
 $ edouttime           : POSIXct[1:546028], format: "2180-05-06 23:30:00" "2180-06-26 21:31:0
 $ hospital_expire_flag: int [1:546028] 0 0 0 0 0 0 0 0 0 0 ...
 - attr(*, "spec")=
  .. 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()
  .. )
 - attr(*, "problems")=<externalptr>
```

```
memory_tidy_optimized
```

```
63.47 MB
```

- New run time: 1.241s
- New memory: 63.47 MB
- The run time is slightly faster (1.241s $<$ 1.432s) and the memory usage also decreased slightly (63.47 MB $<$ 70.02 MB).

Q2. Ingest big data files

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

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

```
-rw-r--r--  1 emma  staff  2592909134 Oct  4 00:08 /Users/emma/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,valu
1,10000032,,2704548,50931,P69FQC,2180-03-23 11:51:00,2180-03-23 15:56:00,___,95,mg/dL,70,100
2,10000032,,36092842,51071,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,ROUTINE,
3,10000032,,36092842,51074,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,ROUTINE,
4,10000032,,36092842,51075,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,ROUTINE,"
5,10000032,,36092842,51079,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,ROUTINE,
6,10000032,,36092842,51087,P69FQC,2180-03-23 11:51:00,,,,,,,ROUTINE,RANDOM.
7,10000032,,36092842,51089,P69FQC,2180-03-23 11:51:00,2180-03-23 16:15:00,,,,,,ROUTINE,PRESU
8,10000032,,36092842,51090,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,ROUTINE,MI
9,10000032,,36092842,51092,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,ROUTINE,"(
```

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

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

```
# labevents <- "~/mimic/hosp/labevents.csv.gz"
# df_labs <- read_csv(labevents)
# str(df_labs)
# head(df_labs)
```

It's taking more than 3 minutes. read_csv() loads the entire dataset into memory, and the file is too large for it to handle. read_csv() also automatically infers column types, which slows down the reading process.

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`.)

```
# labevents <- "~/mimic/hosp/labevents.csv.gz"
# df_labs_selected <- read_csv(labevents, col_select = c(subject_id, itemid, charttime, valu
# str(df_labs_selected)
```

It's still taking very long. This is because read_csv() still reads the whole file line by line, and only keeps the selected columns.

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

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

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

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

```
zcat < labevents_filtered.csv.gz | tail -n +2 | head -10
```

```
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
```

```
zcat < labevents_filtered.csv.gz | tail -n +2 | wc -l
```

```
 32679896
```

Number of lines: 32679896

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

```
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.

   user  system elapsed
 48.711  22.259  13.845
```

It takes 20.356s to ingest.

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.

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

```
library(arrow)
library(dplyr)

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

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

df_filtered <- filtered_data
```

```
system.time({
  df_filtered <- dataset %>%
    select(subject_id, itemid, charttime, valuenum) %>%
    filter(
      itemid %in% c(
        50912, 50971, 50983,
        50902, 50882, 51221,
        51301, 50931
      )
    ) %>%
    collect()
})
```

```
   user  system elapsed
 48.304   9.101  44.058
```

It takes 41.013s to ingest+select+filter.

```
nrow(df_filtered)
```

```
[1] 32679896
```

```
head(df_filtered, 10)
```

```
# A tibble: 10 x 4
   subject_id itemid charttime           valuenum
        <int>  <int> <dttm>                 <dbl>
 1   10000032  50931 2180-03-23 19:51:00     95
 2   10000032  50882 2180-03-23 19:51:00     27
 3   10000032  50902 2180-03-23 19:51:00    101
 4   10000032  50912 2180-03-23 19:51:00      0.4
 5   10000032  50971 2180-03-23 19:51:00      3.7
 6   10000032  50983 2180-03-23 19:51:00    136
 7   10000032  51221 2180-03-23 19:51:00     45.4
 8   10000032  51301 2180-03-23 19:51:00      3
 9   10000032  51221 2180-05-07 06:25:00     42.6
10   10000032  51301 2180-05-07 06:25:00      5
```

Number of rows: 32679896

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

- Apache Arrow can access, process, and move large datasets much faster.
- This is useful for big data, real-time analytics, and machine learning, etc.
- Arrow can also read, write, and process data across different programming languages.

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.)

```
csv_path <- "labevents.csv"
parquet_path <- "labevents.parquet"
dataset <- open_dataset(csv_path, format = "csv")
write_dataset(dataset, parquet_path, format = "parquet")
```

```
file.info(parquet_path)$size
```

```
[1] 96
```

Size: 96B

```
parquet_data <- open_dataset(parquet_path, format = "parquet")

system.time({
  df_parquet_filtered <- parquet_data %>%
    select(subject_id, itemid, charttime, valuenum) %>%
    filter(
      itemid %in% c(
        50912, 50971, 50983,
        50902, 50882, 51221,
        51301, 50931
      )
    ) %>%
    collect()
})
```

```
   user  system elapsed
 20.660   6.951   4.532
```

It takes 5.212s to ingest+select+filter.

```
nrow(df_parquet_filtered)
```

```
[1] 32679896
```

```
head(df_parquet_filtered, 10)
```

```
# A tibble: 10 x 4
   subject_id itemid charttime           valuenum
        <int>  <int> <dttm>                 <dbl>
 1   10000032  50931 2180-03-23 19:51:00     95
 2   10000032  50882 2180-03-23 19:51:00     27
 3   10000032  50902 2180-03-23 19:51:00    101
 4   10000032  50912 2180-03-23 19:51:00      0.4
 5   10000032  50971 2180-03-23 19:51:00      3.7
 6   10000032  50983 2180-03-23 19:51:00    136
 7   10000032  51221 2180-03-23 19:51:00     45.4
 8   10000032  51301 2180-03-23 19:51:00      3
 9   10000032  51221 2180-05-07 06:25:00     42.6
10   10000032  51301 2180-05-07 06:25:00      5
```

Number of rows: 32679896

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

- Parquet arranges data in columns so it can find and retrieve information much faster.
- It's also highly compressed, saving space while keeping things efficient.

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.)

```r
parquet_path <- "labevents.parquet"

# Step 1: Open the Parquet file using Arrow
dataset <- arrow::open_dataset(parquet_path, format = "parquet")

# Step 3: Register the Arrow dataset as a DuckDB table
duckdb_table <- arrow::to_duckdb(dataset, table = "lab_duckdb")
```

```r
system.time({
  filtered_data <- duckdb_table %>%
    select(subject_id, itemid, charttime, valuenum) %>%
    filter(itemid %in% c(50912, 50971, 50983, 50902, 50882, 51221, 51301, 50931))
```

```
  df_duckdb_filtered <- collect(filtered_data)
})
```

```
   user  system elapsed
 22.856   4.523   3.978
```

It takes 4.071s to ingest+convert+select+filter.

```
nrow(df_duckdb_filtered)
```

```
[1] 32679896
```

Number of rows: 32679896

```
head(filtered_data, 10)
```

```
# Source:   SQL [10 x 4]
# Database: DuckDB v1.1.3 [root@Darwin 21.5.0:R 4.4.1/:memory:]
   subject_id itemid charttime           valuenum
        <dbl>  <dbl> <dttm>                 <dbl>
 1   10000032  50931 2180-03-23 11:51:00     95
 2   10000032  50882 2180-03-23 11:51:00     27
 3   10000032  50902 2180-03-23 11:51:00    101
 4   10000032  50912 2180-03-23 11:51:00      0.4
 5   10000032  50971 2180-03-23 11:51:00      3.7
 6   10000032  50983 2180-03-23 11:51:00    136
 7   10000032  51221 2180-03-23 11:51:00     45.4
 8   10000032  51301 2180-03-23 11:51:00      3
 9   10000032  51221 2180-05-06 22:25:00     42.6
10   10000032  51301 2180-05-06 22:25:00      5
```

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

- DuckDB is a database that can store and retrieve data very quickly.
- DuckDB can process billions of rows in seconds without needing a big database server.
- It's optimized for analytics, such as filter, aggregate, and analyze data.

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

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

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

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

How many rows? 433 millions.

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

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

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

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

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

```
library(arrow)
library(dplyr)

csv_file <- "~/mimic/icu/chartevents.csv.gz"
parquet_file <- "~/mimic/icu/csv_file"

dataset <- open_dataset(csv_file, format = "csv")
write_dataset(dataset, parquet_file, format = "parquet")

parquet_data <- open_dataset(parquet_file, format = "parquet")

filtered_data <- parquet_data %>%
  filter(itemid %in% c(220045, 220181, 220179, 223761, 220210))
df_filtered <- collect(filtered_data)
```

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

```
nrow(df_filtered)
```

```
[1] 30195426
```

```
head(df_filtered, 10)
```

```
# A tibble: 10 x 11
   subject_id  hadm_id  stay_id caregiver_id charttime
        <int>    <int>    <int>        <int> <dttm>
 1   10000032 29079034 39553978        18704 2180-07-23 22:00:00
 2   10000032 29079034 39553978        18704 2180-07-23 22:11:00
 3   10000032 29079034 39553978        18704 2180-07-23 22:11:00
 4   10000032 29079034 39553978        18704 2180-07-23 22:12:00
 5   10000032 29079034 39553978        18704 2180-07-23 22:12:00
 6   10000032 29079034 39553978        18704 2180-07-23 22:30:00
 7   10000032 29079034 39553978        18704 2180-07-23 22:30:00
 8   10000032 29079034 39553978        18704 2180-07-23 22:30:00
```

```
 9   10000032 29079034 39553978        18704 2180-07-23 22:30:00
10   10000032 29079034 39553978        18704 2180-07-23 23:00:00
# i 6 more variables: storetime <dttm>, itemid <int>, value <chr>,
#   valuenum <dbl>, valueuom <chr>, warning <int>
```

Number of rows: 30195426