# Biostat 203B Homework 2

**Due Feb 7, 2025 @ 11:59PM**

Yanzi Sun 106183069

## Table of contents



Display machine information for reproducibility:

```
sessionInfo()
```

```
R version 4.4.2 (2024-10-31)
Platform: aarch64-apple-darwin20
Running under: macOS Sonoma 14.7.1

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib;  L

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: America/Los_Angeles
```

```
tzcode source: internal

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

loaded via a namespace (and not attached):
 [1] compiler_4.4.2    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
```

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

```
Loading required package: R.oo
```

```
Loading required package: R.methodsS3
```

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

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

```

Attaching package: 'R.oo'
```

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

    throw
```

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

    getClasses, getMethods
```

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

    attach, detach, load, save
```

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

```

Attaching package: 'R.utils'
```

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

    timestamp
```

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

    timestamp
```

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

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

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.4     v tidyr     1.3.1
v purrr     1.0.2

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

Display memory information of your computer

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

```
Totalram:  16.000 GiB
Freeram:    1.749 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 12306248
-rw-r--r--@ 1 yanzisun  staff    19928140 Jun 24  2024 admissions.csv.gz
-rw-r--r--@ 1 yanzisun  staff      427554 Apr 12  2024 d_hcpcs.csv.gz
-rw-r--r--@ 1 yanzisun  staff      876360 Apr 12  2024 d_icd_diagnoses.csv.gz
-rw-r--r--@ 1 yanzisun  staff      589186 Apr 12  2024 d_icd_procedures.csv.gz
-rw-r--r--@ 1 yanzisun  staff       13169 Oct  3 09:07 d_labitems.csv.gz
-rw-r--r--@ 1 yanzisun  staff    33564802 Oct  3 09:07 diagnoses_icd.csv.gz
-rw-r--r--@ 1 yanzisun  staff     9743908 Oct  3 09:07 drgcodes.csv.gz
-rw-r--r--@ 1 yanzisun  staff   811305629 Apr 12  2024 emar.csv.gz
-rw-r--r--@ 1 yanzisun  staff   748158322 Apr 12  2024 emar_detail.csv.gz
-rw-r--r--@ 1 yanzisun  staff     2162335 Apr 12  2024 hcpcsevents.csv.gz
-rw-r--r--@ 1 yanzisun  staff  2592909134 Oct  3 09:08 labevents.csv.gz
-rw-r--r--@ 1 yanzisun  staff   117644075 Oct  3 09:08 microbiologyevents.csv.gz
-rw-r--r--@ 1 yanzisun  staff    44069351 Oct  3 09:08 omr.csv.gz
-rw-r--r--@ 1 yanzisun  staff     2835586 Apr 12  2024 patients.csv.gz
-rw-r--r--@ 1 yanzisun  staff   525708076 Apr 12  2024 pharmacy.csv.gz
-rw-r--r--@ 1 yanzisun  staff   666594177 Apr 12  2024 poe.csv.gz
-rw-r--r--@ 1 yanzisun  staff    55267894 Apr 12  2024 poe_detail.csv.gz
-rw-r--r--@ 1 yanzisun  staff   606298611 Apr 12  2024 prescriptions.csv.gz
-rw-r--r--@ 1 yanzisun  staff     7777324 Apr 12  2024 procedures_icd.csv.gz
-rw-r--r--@ 1 yanzisun  staff      127330 Apr 12  2024 provider.csv.gz
-rw-r--r--@ 1 yanzisun  staff     8569241 Apr 12  2024 services.csv.gz
-rw-r--r--@ 1 yanzisun  staff    46185771 Oct  3 09:08 transfers.csv.gz
```

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

```
total 8506784
-rw-r--r--@ 1 yanzisun  staff       41566 Apr 12  2024 caregiver.csv.gz
```

```
-rw-r--r--@ 1 yanzisun  staff  3502392765 Apr 12  2024 chartevents.csv.gz
-rw-r--r--@ 1 yanzisun  staff       58741 Apr 12  2024 d_items.csv.gz
-rw-r--r--@ 1 yanzisun  staff    63481196 Apr 12  2024 datetimeevents.csv.gz
-rw-r--r--@ 1 yanzisun  staff     3342355 Oct  3 07:36 icustays.csv.gz
-rw-r--r--@ 1 yanzisun  staff   311642048 Apr 12  2024 ingredientevents.csv.gz
-rw-r--r--@ 1 yanzisun  staff   401088206 Apr 12  2024 inputevents.csv.gz
-rw-r--r--@ 1 yanzisun  staff    49307639 Apr 12  2024 outputevents.csv.gz
-rw-r--r--@ 1 yanzisun  staff    24096834 Apr 12  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.

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:** The fread function is the fastest.

I used str() to check the parsed data types and found: read.csv in base R creates a data.frame; read_csv in tidyverse creates a tibble, and fread in the data.table package creates a data.table which also behaves like data.frame.

The memory use for each function is shown below.

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

```
   user  system elapsed 
  6.122   0.056   6.181 
```

```
system.time(tidyverse_read <- read_csv("~/mimic/hosp/admissions.csv.gz"))
```

```
Rows: 546028 Columns: 16
-- Column specification --------------------------------------------------------
Delimiter: ","
chr  (8): admission_type, admit_provider_id, admission_location, discharge_l...
dbl  (3): subject_id, hadm_id, hospital_expire_flag
```

```
dttm (5): admittime, dischtime, deathtime, edregtime, edouttime

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

```
   user  system elapsed
  1.156   0.092   0.701
```

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

```
   user  system elapsed
  0.688   0.095   0.368
```

```
object_size(base_read)
```

```
200.10 MB
```

```
object_size(tidyverse_read)
```

```
70.02 MB
```

```
object_size(fread_read)
```

```
63.47 MB
```

```
str(base_read)
```

```
'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 ...
```

```
str(tidyverse_read)
```

```
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>
```

```
str(fread_read)
```

```
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>
```

### 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:** The run time is almost the same as me not indicating column data types. And the memory is exactly the same as just using read_csv.

```
system.time(specified_column<-read_csv("~/mimic/hosp/admissions.csv.gz",
                                       col_types = NULL))
```

```
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
  1.085   0.079   0.569
```

```
object_size(specified_column)
```

```
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 yanzisun  staff  2592909134 Oct  3 09:08 /Users/yanzisun/mimic/hosp/labevents.c
```

Display the first 10 lines of this file.

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

```
labevent_id,subject_id,hadm_id,specimen_id,itemid,order_provider_id,charttime,storetime,value
1,10000032,,2704548,50931,P69FQC,2180-03-23 11:51:00,2180-03-23 15:56:00,___,95,mg/dL,70,100
2,10000032,,36092842,51071,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,ROUTINE,
3,10000032,,36092842,51074,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,ROUTINE,
4,10000032,,36092842,51075,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,ROUTINE,"I
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.

**Solution:** My R crashes. The reason is the file is too huge once decompressed. The reading process is taking more memory than my computer's capability(8gb ram).

```
read_csv("~/mimic/hosp/labevents.csv.gz")
```

### 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:** No, it does not solve the ingestion issue, still taking forever to run chunk. This is probably due to csv files are read row by rows. Selecting columns cannot shorten the time of reading rows.

```
system.time(read_csv("~/mimic/hosp/labevents.csv.gz",
                     col_select= c("subject_id", "itemid",
                                   "charttime", "valuenum")))
```

### 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:** There are 32679896 lines in the new filtered file excluding the header. It takes the system around 0.5 seconds for `read_csv` to ingest the filtered csv.gz.(32 seconds for user)

```
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) {
    print $2 "," $5 "," $7 "," $10
}' | gzip > labevents_filtered.csv.gz
```

```
zcat < 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
```

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

```
zcat: (stdin): unexpected end of file
 5360461
```

```
system.time(read.csv("labevents_filtered.csv.gz"))
```

```
   user  system elapsed 
  5.425   0.087   5.516 
```

### 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:** It takes 3.69 seconds of system time for apache arrow to finish the entire process. Apache Arrow is a tool that makes working with big data super fast and simple. Imagine you have a huge file, like a giant table, and instead of making copies or waiting a long time to load it, Arrow lets different programs—like Python, R, or databases—read it directly and quickly. It saves memory and speeds up tasks, like analyzing data or running models. It provides a smart way for different tools to share and work with the same data without wasting time or resources.

```
system.time({
  apache_tibble <- arrow::open_dataset("labevents.csv", format = "csv") %>%
  filter(itemid %in% c(50912, 50971, 50983, 50902,
                       50882, 51221, 51301, 50931)) %>%
  select(subject_id, itemid, charttime, valuenum) %>%
  arrange(subject_id, charttime, itemid) %>%
  collect()})
```

```
   user  system elapsed
 46.254   3.537  44.272
```

```
head(apache_tibble)
```

```
# A tibble: 6 x 4
  subject_id itemid charttime           valuenum
       <int>  <int> <dttm>                 <dbl>
1   10000032  50882 2180-03-23 04:51:00     27
2   10000032  50902 2180-03-23 04:51:00    101
3   10000032  50912 2180-03-23 04:51:00      0.4
4   10000032  50931 2180-03-23 04:51:00     95
5   10000032  50971 2180-03-23 04:51:00      3.7
6   10000032  50983 2180-03-23 04:51:00    136
```

```
nrow(apache_tibble)
```

```
[1] 32679896
```

### 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:** The file size is 2.5 GB, and the system time used by the Parquet file is shown below. The row number matches that of Q2.3. Parquet is a way to store data that's smart and efficient. Instead of saving data row by row like a regular file (like CSV), it saves it column by column. This makes it faster to work with, especially if you only need specific parts of the data. Plus, it compresses the data, so it takes up less space on your computer. Think of it like packing your suitcase neatly to fit more stuff while keeping it easy to find what you need. It's great for handling big datasets quickly.

```
system.time({
  arrow::write_dataset(
    open_dataset("labevents.csv", format = "csv"),
    path = "labevents_parquet",
    format = "parquet"
  )
})
```

```
   user  system elapsed
109.947   6.322  59.308
```

```
system.time({
  parquet_tibble <- arrow::open_dataset("labevents_parquet",
                                        format = "parquet") %>%
    filter(itemid %in% c(50912, 50971, 50983, 50902, 50882,
                51221, 51301, 50931)) %>%
    select(subject_id, itemid, charttime, valuenum) %>%
    arrange(subject_id, charttime, itemid) %>%
    collect()
})
```

```
   user  system elapsed
 12.890   2.855   5.877
```

```
head(parquet_tibble, 10)
```

```
# A tibble: 10 x 4
   subject_id itemid charttime           valuenum
        <int>  <int> <dttm>                 <dbl>
 1   10000032  50882 2180-03-23 04:51:00     27
 2   10000032  50902 2180-03-23 04:51:00    101
 3   10000032  50912 2180-03-23 04:51:00      0.4
 4   10000032  50931 2180-03-23 04:51:00     95
 5   10000032  50971 2180-03-23 04:51:00      3.7
 6   10000032  50983 2180-03-23 04:51:00    136
 7   10000032  51221 2180-03-23 04:51:00     45.4
 8   10000032  51301 2180-03-23 04:51:00      3
 9   10000032  50882 2180-05-06 15:25:00     27
10   10000032  50902 2180-05-06 15:25:00    105
```

```
nrow(parquet_tibble)
```

```
[1] 32679896
```

### 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:** It takes around 2 seconds for the system to process the entire workflow. DuckDB is a super-fast, lightweight database designed for data analysis. It's great for working with large files like CSVs or Parquet, and it can handle millions of rows quickly without using too much memory. DuckDB helps you filter, sort, and analyze data much faster than using traditional data frames. It's like SQLite for analytics, but designed to crunch numbers at lightning speed.

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

```
  duckdb_connection <- dbConnect(duckdb::duckdb())
  duckdb_table <- arrow::to_duckdb(parquet_dataset, con = duckdb_connection)

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

```
   user  system elapsed
 18.207   9.406   7.415
```

```
head(duckdb_tibble, 10)
```

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

```
nrow(duckdb_tibble)
```

```
[1] 32679896
```

```
dbDisconnect(duckdb_connection)
```

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

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

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

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

How many rows? 433 millions.

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

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

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

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

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

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

**Solution:** I choose to use apache arrow with dplyr to answer this question. First, I downloaded the unzipped chartevnts.csv from box folder, then I ingest it with open_dataset function and output a parquet file. Next, I created a tibble collecting the dplyer-filtered dataset. Finally, I showed the nrow and head the first 10 rows. Please remove # since I made a parquet file in my memory and rendering takes too much memory.

```
#arrow::write_dataset(
    #open_dataset("chartevents.csv", format = "csv"),
    #path = "chartevents_parquet",
    #format = "parquet")

chartevents_tibble <- arrow::open_dataset("chartevents_parquet",
                                          format = "parquet") %>%
    filter(itemid %in% c(220045, 220181, 220179, 223761, 220210)) %>%
  arrange(subject_id, charttime, itemid) %>%
    collect()

head(chartevents_tibble, 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 07:00:00
 2   10000032 29079034 39553978        18704 2180-07-23 07:11:00
 3   10000032 29079034 39553978        18704 2180-07-23 07:11:00
 4   10000032 29079034 39553978        18704 2180-07-23 07:12:00
 5   10000032 29079034 39553978        18704 2180-07-23 07:12:00
 6   10000032 29079034 39553978        18704 2180-07-23 07:30:00
 7   10000032 29079034 39553978        18704 2180-07-23 07:30:00
 8   10000032 29079034 39553978        18704 2180-07-23 07:30:00
 9   10000032 29079034 39553978        18704 2180-07-23 07:30:00
10   10000032 29079034 39553978        18704 2180-07-23 08:00:00
# i 6 more variables: storetime <dttm>, itemid <int>, value <chr>,
#   valuenum <dbl>, valueuom <chr>, warning <int>
```

```
nrow(chartevents_tibble)
```

```
[1] 30195426
```