Biostat 203B Homework 2

Due Feb 7, 2025 @ 11:59PM

AUTHOR
Zhiyuan Yu 906405523

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] htmlwidgets_1.6.4 compiler_4.4.2    fastmap_1.2.0     cli_3.6.3
 [5] tools_4.4.2       htmltools_0.5.8.1 rstudioapi_0.17.1 yaml_2.3.10
 [9] rmarkdown_2.29    knitr_1.49        jsonlite_1.8.9    xfun_0.50
[13] digest_0.6.37     rlang_1.1.4       evaluate_1.0.3
```

getOption("pkgType")

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 ──
✓ 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
```

```
library(stringr)
```

Display memory information of your computer

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

```
Totalram:  15.463 GiB
Freeram:   13.993 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 24124660
-rwxrwxrwx 1 zxhyu zxhyu    19928140 Jan 21 23:40 admissions.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu      427554 Jan 21 23:40 d_hcpcs.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu      876360 Jan 21 23:40 d_icd_diagnoses.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu      589186 Jan 21 23:40 d_icd_procedures.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu       13169 Jan 21 23:40 d_labitems.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu    33564802 Jan 21 23:40 diagnoses_icd.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu     9743908 Jan 21 23:40 drgcodes.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu   811305629 Jan 21 23:40 emar.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu   748158322 Jan 21 23:40 emar_detail.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu     2162335 Jan 21 23:40 hcpcsevents.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu 18402851720 Jan 21 23:41 labevents.csv
-rwxrwxrwx 1 zxhyu zxhyu  2592909134 Jan 21 23:41 labevents.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu   117644075 Jan 21 23:41 microbiologyevents.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu    44069351 Jan 21 23:41 omr.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu     2835586 Jan 21 23:41 patients.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu   525708076 Jan 21 23:41 pharmacy.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu   666594177 Jan 21 23:41 poe.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu    55267894 Jan 21 23:41 poe_detail.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu   606298611 Jan 21 23:41 prescriptions.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu     7777324 Jan 21 23:41 procedures_icd.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu      127330 Jan 21 23:41 provider.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu     8569241 Jan 21 23:41 services.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu    46185771 Jan 21 23:41 transfers.csv.gz
```

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

```
total 45206328
-rwxrwxrwx 1 zxhyu zxhyu       41566 Jan 21 23:41 caregiver.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu 41935806083 Jan 21 23:42 chartevents.csv
-rwxrwxrwx 1 zxhyu zxhyu  3502392765 Jan 21 23:42 chartevents.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu       58741 Jan 21 23:42 d_items.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu    63481196 Jan 21 23:42 datetimeevents.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu     3342355 Jan 21 23:42 icustays.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu   311642048 Jan 21 23:42 ingredientevents.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu   401088206 Jan 21 23:42 inputevents.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu    49307639 Jan 21 23:42 outputevents.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu    24096834 Jan 21 23:42 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:**

```
# Determine the file path
file_path_1 <- "~/mimic/hosp/admissions.csv.gz"

# Read with read.csv in base R
time_read_csv_base <- system.time({
  df_base <- read.csv(file_path_1)
})

# Read with read_csv in tidyverse
time_read_csv_tidyverse <- system.time({
  df_tidyverse <- read_csv(file_path_1)
})
```

```
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.
```

```
# Read with fread in data.table package
time_fread_datatable <- system.time({
  df_datatable <- fread(file_path_1)
})

# Showing and Compare the Data Types
cat("Base R Data Types:\n")
```

```
Base R Data Types:
```

```
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 ...
```

```
cat("\nTidyverse Data Types:\n")
```

```
Tidyverse Data Types:
```

```
str(df_tidyverse)
```

```
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>
```

```
cat("\nData.table Data Types:\n")
```

```
Data.table Data Types:
```

```
str(df_datatable)
```

```
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>
```

```
# Memory Usage by each method
memory_base <- object_size(df_base)
memory_tidyverse <- object_size(df_tidyverse)
memory_datatable <- object_size(df_datatable)

# Displaying Results
results <- data.frame(
  Method = c("read.csv (base R)", "read_csv (tidyverse)", "fread (data.table)"),
  Time_in_seconds = c(time_read_csv_base["elapsed"],
                      time_read_csv_tidyverse["elapsed"],
                      time_fread_datatable["elapsed"]),
  Memory_Usage_MB = c(as.numeric(memory_base) / (1024^2),
                      as.numeric(memory_tidyverse) / (1024^2),
                      as.numeric(memory_datatable) / (1024^2))
)

print(results)
```

```
               Method Time_in_seconds Memory_Usage_MB
1    read.csv (base R)          13.411       190.82912
2 read_csv (tidyverse)           2.815        66.77875
3   fread (data.table)           1.773        60.52495
```

As shown above, fread in the data.table package is the fastest while read.csv in base R is the slowest. The memory taken up by each resultant dataframe or tibble use are also shown above. For default parsed data types, while read_csv and fread are similar except for different naming for numeric data (`int` for fread and `num` for read_csv), read.csv is different in some variables' data type such as catigorizing time as chr while the other two methods catigorize as POSIXct.

Q1.2 User-supplied data types

Re-ingest `admissions.csv.gz` by indicating appropriate column data types in `read_csv` . Does the run time change? How much memory does the result tibble use? (Hint: `col_types` argument in `read_csv` .) **Solution:**

```
# File path
file_path_2 <- "~/mimic/hosp/admissions.csv.gz"

# Specified column types based on the provided structure
col_types_specified <- 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()
)

# Measure run time with specified column types
time_with_col_types <- system.time({
  df_col_types <- read_csv(file_path_2, col_types = col_types_specified)
})

# Measure memory usage
memory_col_types <- object_size(df_col_types)

# Displaying Results
results_with_col_types <- data.frame(
  Method = "read_csv (specified col_types)",
  Time_in_seconds = time_with_col_types["elapsed"],
  Memory_Usage_MB = as.numeric(memory_col_types) / (1024^2)
)

print(results_with_col_types)
```

```
                              Method Time_in_seconds Memory_Usage_MB
elapsed read_csv (specified col_types)            2.49        66.77896
```

The run time changes. The run time and the memory usage of the result tibble are shown above.

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 zxhyu zxhyu 2592909134 Jan 21 23:41 /home/zxhyu/mimic/hosp/labevents.csv.gz
```

Display the first 10 lines of this file.

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

```
labevent_id,subject_id,hadm_id,specimen_id,itemid,order_provider_id,charttime,storetime,value,val
uenum,valueuom,ref_range_lower,ref_range_upper,flag,priority,comments
1,10000032,,2704548,50931,P69FQC,2180-03-23 11:51:00,2180-03-23
15:56:00,___,95,mg/dL,70,100,,ROUTINE,"IF FASTING, 70-100 NORMAL, >125 PROVISIONAL DIABETES."
2,10000032,,36092842,51071,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,,ROUTINE,
3,10000032,,36092842,51074,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,,ROUTINE,
4,10000032,,36092842,51075,P69FQC,2180-03-23 11:51:00,2180-03-23
16:00:00,NEG,,,,,,ROUTINE,"BENZODIAZEPINE IMMUNOASSAY SCREEN DOES NOT DETECT SOME
DRUGS,;INCLUDING LORAZEPAM, CLONAZEPAM, AND FLUNITRAZEPAM."
5,10000032,,36092842,51079,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,,ROUTINE,
6,10000032,,36092842,51087,P69FQC,2180-03-23 11:51:00,,,,,,,,ROUTINE,RANDOM.
7,10000032,,36092842,51089,P69FQC,2180-03-23 11:51:00,2180-03-23
16:15:00,,,,,,,ROUTINE,PRESUMPTIVELY POSITIVE.
8,10000032,,36092842,51090,P69FQC,2180-03-23 11:51:00,2180-03-23
16:00:00,NEG,,,,,,ROUTINE,METHADONE ASSAY DETECTS ONLY METHADONE (NOT OTHER OPIATES/OPIOIDS).
9,10000032,,36092842,51092,P69FQC,2180-03-23 11:51:00,2180-03-23
16:00:00,NEG,,,,,,ROUTINE,"OPIATE IMMUNOASSAY SCREEN DOES NOT DETECT SYNTHETIC OPIOIDS;SUCH AS
METHADONE, OXYCODONE, FENTANYL, BUPRENORPHINE, TRAMADOL,;NALOXONE, MEPERIDINE.  SEE ONLINE LAB
MANUAL FOR DETAILS."
```

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

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

```
# Set a timer to monitor the performance
start_time <- Sys.time()

# Use tryCatch to handle interruptions
result <- tryCatch({
  df_readr <- read_csv("~/mimic/hosp/labevents.csv.gz")
  end_time <- Sys.time()

  # Calculate duration
  duration <- as.numeric(difftime(end_time, start_time, units = "secs"))

  cat("Ingestion completed in", duration, "seconds.\n")
  cat("Data dimensions:", dim(df_readr), "\n")

  return(df_readr)
}, error = function(e) {
  cat("An error occurred:", e$message, "\n")
  return(NULL)
})
```

It took my system more than 3 minutes before it crashed and the process of ingestion did not finish. The reason for this situation was because the file was too large and my laptop does not have enough memory (RAM) to process such large file.

Q2.2 Ingest selected columns of `labevents.csv.gz` by `read_csv`

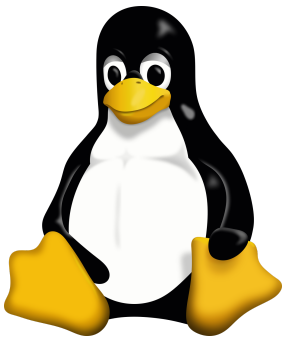
Try to ingest only columns `subject_id`, `itemid`, `charttime`, and `valuenum` in `labevents.csv.gz` using `read_csv`. Does this solve the ingestion issue? (Hint: `col_select` argument in `read_csv`.) **Solution:**

```
# Measure the time for ingestion
start_time <- Sys.time()

# Use tryCatch to handle any potential errors
result <- tryCatch({
  # Ingest only selected columns
```

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

  end_time <- Sys.time()

  # Calculate duration
  duration <- as.numeric(difftime(end_time, start_time, units = "secs"))

  # Display performance and data information
  cat("Ingestion completed in", duration, "seconds.\n")
  cat("Data dimensions:", dim(df_selected), "\n")

  return(df_selected)
}, error = function(e) {
  cat("An error occurred:", e$message, "\n")
  return(NULL)
})
```

My program was forced to terminate again since the file was still to large and my laptop did not have enough RAM to ingest the data.

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



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

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

```
# Show the headers
zcat ~/mimic/hosp/labevents.csv.gz | head -n 1
```

```
# Extract the wanted columns and rows and put them into a new file
zcat < ~/mimic/hosp/labevents.csv.gz |
  awk -F, 'BEGIN {OFS=","; print "subject_id, itemid, charttime, valuenum"}
  NR==1 {next}
  ($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
```

The file is created

Display the first 10 lines of the new file `labevents_filtered.csv.gz`. How many lines are in this new file, excluding the header? How long does it take `read_csv` to ingest `labevents_filtered.csv.gz`?

Solution: Display the first 10 lines of the new file excluding header:

```
zcat labevents_filtered.csv.gz | head -n 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 lines in the new file:

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

```
32679896
```

Measure the time it takes `read_csv` to ingest the new file:

```
time Rscript -e "read.csv('labevents_filtered.csv.gz')" > /dev/null 2>&1
```

```
real    0m57.656s
user    0m56.225s
sys 0m1.010s
```

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: Step 1, unzip the labevents.csv.gz file

```
gunzip -k ~/mimic/hosp/labevents.csv.gz
```

Step 2, measure the time for ingest + select + filter

```
time_arrow <- system.time({
  dataset <- arrow::open_dataset("~/mimic/hosp/labevents.csv", format = "csv")

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

print(time_arrow)
```

```
   user  system elapsed 
 64.485   9.007 275.465 
```

Step 3, count the number of rows:

```
cat("Number of Rows:", nrow(result_tibble), "\n")
```

```
Number of Rows: 32679896
```

Step 4, show the first 10 rows:

```
print(head(result_tibble, 10))
```

```
# A tibble: 10 × 4
   subject_id itemid charttime           valuenum
        <int>  <int> <dttm>                 <dbl>
 1   10000032  50931 2180-03-23 04:51:00     95
 2   10000032  50882 2180-03-23 04:51:00     27
 3   10000032  50902 2180-03-23 04:51:00    101
 4   10000032  50912 2180-03-23 04:51:00      0.4
 5   10000032  50971 2180-03-23 04:51:00      3.7
 6   10000032  50983 2180-03-23 04:51:00    136
 7   10000032  51221 2180-03-23 04:51:00     45.4
 8   10000032  51301 2180-03-23 04:51:00      3
 9   10000032  51221 2180-05-06 15:25:00     42.6
10   10000032  51301 2180-05-06 15:25:00      5
```

Apache Arrow is a method to move and process large amounts of data within a short time. Imagine you bought a large bag of rice from Costco and tried to pour it into a rice bucket. If you are not strong enough to carry the bag and pour the rice directly, you will need to transfer the rice with a small rice cup little by little to the bucket, which could take a long time. Apache Arrow is like a secret strength booster which lets you move the entire bag of rice at once and pour it into the bucket, saving you a lot of time and trouble. Apache Arrow stores data in a format that different tools and programs can understand instantly, making data analysis quicker more efficient, especially when dealing with big files that require insant amount of computer memory.

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:** Step 1, re-write the csv file into binary Parquet format:

```
labevents_data <- arrow::open_dataset("~/mimic/hosp/labevents.csv", format = "csv")

arrow::write_dataset(labevents_data, path = "labevents_parquet", format = "parquet")
```

Step 2, measure the size of the file:

```
du -sh labevents_parquet
```

```
2.6G    labevents_parquet
```

Step 3, measure the time for ingest + select + filter

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

result_tibble_parquet <- dataset_parquet %>%
  select (subject_id, itemid, charttime, valuenum) %>%
  filter(itemid %in% c(50912, 50971, 50983, 50902, 50882, 51221,
                       51301, 50931)) %>%
  arrange(subject_id, charttime) %>%  # Ensure sorting matches Q2.3
  collect()
})

print(time_parquet)
```

```
   user  system elapsed 
 24.472   9.208  10.230 
```

Step 4: display the number of rows:

```
cat("Number of Rows:", nrow(result_tibble_parquet), "\n")
```

```
Number of Rows: 32679896
```

Step 5, display the first 10 rows:

```
print(head(result_tibble_parquet, 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 
```

Parquet is a method that allow people to efficiently store large datasets. It functions like a higly organized, compressed data package which allow users to process the data within a short time. It's a useful tool for processing large amount of data since it saves space and speeds up data processing.

Q2.6 DuckDB

DuckDB

Ingest the Parquet file, convert it to a DuckDB table by `arrow::to_duckdb`, select columns, and filter rows as in Q2.5. How long does the ingest+convert+select+filter process take? Display the number of rows and the first 10 rows of the result tibble and make sure they match those in Q2.3. (Hint: use `dplyr` verbs for selecting columns and filtering rows.)

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

Solution: Step 1, ingest the Parquet file and convert it to a DuckDB table, and measure the time it takes to finish the process:

```
parquet_dir <- "labevents_parquet"

process_time <- system.time({
  # Ingest the Parquet dataset
  dataset_parquet <- arrow::open_dataset(parquet_dir, format = "parquet")

  # Convert the Arrow dataset to a DuckDB table
  con <- dbConnect(duckdb::duckdb())
  duckdb_table <- arrow::to_duckdb(dataset_parquet, con = con, table_name = "labevents_table")

  # Select and filter data
  result_tibble_duckdb <- duckdb_table %>%
    select(subject_id, itemid, charttime, valuenum) %>%
    filter(itemid %in% c(50912, 50971, 50983, 50902, 50882, 51221,
                         51301, 50931)) %>%
    arrange(subject_id, charttime) %>%  # Ensure the same order as Q2.5
    collect()
})

cat("\nIngest + Convert + Select + Filter Time:\n")
```

```
Ingest + Convert + Select + Filter Time:
```

```
print(process_time)
```

```
   user  system elapsed
 62.468  30.078  11.896
```

Step 2, count the number of rows:

```
cat("\nNumber of Rows:", nrow(result_tibble_duckdb), "\n")
```

```
Number of Rows: 32679896
```

Step 3, show the first 10 rows:

```
cat("\nFirst 10 Rows:\n")
```

```
First 10 Rows:
```

```
print(head(result_tibble_duckdb, 10))
```

```
# A tibble: 10 × 4
   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
```

```
# Close DuckDB connection
dbDisconnect(con, shutdown = TRUE)
```

DuckDB is like a lightning-fast, in-memory database that works directly inside your data tools such as R or Python. Consider it as an advanced version of Excel that can handle millions of rows without needing for a big database server. It's great for doing fast data analysis on laptops and this natural of it makes DuckDB a perfect tool for data scientists to work on large datasets.

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
```

```
432997491
```

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: Step 1: Unzip the file

```
gzip -d -k ~/mimic/icu/chartevents.csv.gz"
```

Step 2: Use arrow::open_dataset to ingest + select + filter the file and show the process time

```
time_arrow <- system.time({
  dataset <- arrow::open_dataset("~/mimic/icu/chartevents.csv", format = "csv")

  result_tibble <- dataset %>%
    select(subject_id, itemid, charttime, valuenum) %>%
    filter(itemid %in% c(220045, 220181, 220179, 223761, 220210)) %>%
    collect()
```

```
})

print(time_arrow)
```

```
   user  system elapsed 
137.054  22.029 743.486 
```

Step 3: count the number of rows:

```
cat("Number of Rows:", nrow(result_tibble), "\n")
```

```
Number of Rows: 30195426
```

Step 4: show the first 10 rows:

```
print(head(result_tibble, 10))
```

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