Biostat 203B Homework 2

Due Feb 7, 2025 @ 11:59PM

Jiaye Tian UID: 306541095

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 14.7.3

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_1.1.5       evaluate_1.0.3
```

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

```
library(arrow)
```

```
Attaching package: 'arrow'

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

    timestamp
```

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

```
Loading required package: DBI
```

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

```
Attaching package: 'pryr'

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

    address
```

```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     1.0.4

-- 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:   764.781 MiB
```

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 12306256
-rw-r--r--@ 1 justina  staff    19928140 Jun 24  2024 admissions.csv.gz
-rw-r--r--@ 1 justina  staff      427554 Apr 12  2024 d_hcpcs.csv.gz
-rw-r--r--@ 1 justina  staff      876360 Apr 12  2024 d_icd_diagnoses.csv.gz
-rw-r--r--@ 1 justina  staff      589186 Apr 12  2024 d_icd_procedures.csv.gz
-rw-r--r--@ 1 justina  staff       13169 Oct  3 06:07 d_labitems.csv.gz
-rw-r--r--@ 1 justina  staff    33564802 Oct  3 06:07 diagnoses_icd.csv.gz
-rw-r--r--@ 1 justina  staff     9743908 Oct  3 06:07 drgcodes.csv.gz
-rw-r--r--@ 1 justina  staff   811305629 Apr 12  2024 emar.csv.gz
-rw-r--r--@ 1 justina  staff   748158322 Apr 12  2024 emar_detail.csv.gz
-rw-r--r--@ 1 justina  staff     2162335 Apr 12  2024 hcpcsevents.csv.gz
-rw-r--r--@ 1 justina  staff        2907 Feb  7 23:14 index.html
-rw-r--r--@ 1 justina  staff  2592909134 Oct  3 06:08 labevents.csv.gz
-rw-r--r--@ 1 justina  staff   117644075 Oct  3 06:08 microbiologyevents.csv.gz
-rw-r--r--@ 1 justina  staff    44069351 Oct  3 06:08 omr.csv.gz
-rw-r--r--@ 1 justina  staff     2835586 Apr 12  2024 patients.csv.gz
-rw-r--r--@ 1 justina  staff   525708076 Apr 12  2024 pharmacy.csv.gz
-rw-r--r--@ 1 justina  staff   666594177 Apr 12  2024 poe.csv.gz
-rw-r--r--@ 1 justina  staff    55267894 Apr 12  2024 poe_detail.csv.gz
-rw-r--r--@ 1 justina  staff   606298611 Apr 12  2024 prescriptions.csv.gz
-rw-r--r--@ 1 justina  staff     7777324 Apr 12  2024 procedures_icd.csv.gz
-rw-r--r--@ 1 justina  staff      127330 Apr 12  2024 provider.csv.gz
-rw-r--r--@ 1 justina  staff     8569241 Apr 12  2024 services.csv.gz
-rw-r--r--@ 1 justina  staff    46185771 Oct  3 06:08 transfers.csv.gz
```

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

```
total 8506792
-rw-r--r--@ 1 justina  staff       41566 Apr 12  2024 caregiver.csv.gz
-rw-r--r--@ 1 justina  staff  3502392765 Apr 12  2024 chartevents.csv.gz
-rw-r--r--@ 1 justina  staff       58741 Apr 12  2024 d_items.csv.gz
-rw-r--r--@ 1 justina  staff    63481196 Apr 12  2024 datetimeevents.csv.gz
-rw-r--r--@ 1 justina  staff     3342355 Oct  3 04:36 icustays.csv.gz
-rw-r--r--@ 1 justina  staff        1336 Feb  7 23:14 index.html
-rw-r--r--@ 1 justina  staff   311642048 Apr 12  2024 ingredientevents.csv.gz
-rw-r--r--@ 1 justina  staff   401088206 Apr 12  2024 inputevents.csv.gz
-rw-r--r--@ 1 justina  staff    49307639 Apr 12  2024 outputevents.csv.gz
-rw-r--r--@ 1 justina  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.

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

# `read.csv` (base R) speed, memory, and data types
baseR <- system.time({
  df_base <- read.csv(gzfile(path))
})
mem_base <- pryr::object_size(df_base)
glimpse(df_base)
```

```
Rows: 546,028
Columns: 16
$ subject_id          <int> 10000032, 10000032, 10000032, 10000032, 10000068,~
$ hadm_id             <int> 22595853, 22841357, 25742920, 29079034, 25022803,~
$ admittime           <chr> "2180-05-06 22:23:00", "2180-06-26 18:27:00", "21~
$ dischtime           <chr> "2180-05-07 17:15:00", "2180-06-27 18:49:00", "21~
$ deathtime           <chr> "", "", "", "", "", "", "", "", "", "", "", "", "~
$ admission_type      <chr> "URGENT", "EW EMER.", "EW EMER.", "EW EMER.", "EU~
$ admit_provider_id   <chr> "P49AFC", "P784FA", "P19UTS", "P06OTX", "P39NWO",~
$ admission_location  <chr> "TRANSFER FROM HOSPITAL", "EMERGENCY ROOM", "EMER~
$ discharge_location  <chr> "HOME", "HOME", "HOSPICE", "HOME", "", "HOME HEAL~
$ insurance           <chr> "Medicaid", "Medicaid", "Medicaid", "Medicaid", "~
```

```
$ language             <chr> "English", "English", "English", "English", "Engl~
$ marital_status       <chr> "WIDOWED", "WIDOWED", "WIDOWED", "WIDOWED", "SING~
$ race                 <chr> "WHITE", "WHITE", "WHITE", "WHITE", "WHITE", "WHI~
$ edregtime            <chr> "2180-05-06 19:17:00", "2180-06-26 15:54:00", "21~
$ edouttime            <chr> "2180-05-06 23:30:00", "2180-06-26 21:31:00", "21~
$ hospital_expire_flag <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
```

```
# `read_csv` (tidyverse) speed, memory, and data types
tidyverse <- system.time({
  df_tidyverse <- read_csv(gzfile(path))
})
```

```
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.
```

```
mem_tidyverse <- pryr::object_size(df_tidyverse)
glimpse(df_tidyverse)
```

```
Rows: 546,028
Columns: 16
$ subject_id           <dbl> 10000032, 10000032, 10000032, 10000032, 10000068,~
$ hadm_id              <dbl> 22595853, 22841357, 25742920, 29079034, 25022803,~
$ admittime            <dttm> 2180-05-06 22:23:00, 2180-06-26 18:27:00, 2180-0~
$ dischtime            <dttm> 2180-05-07 17:15:00, 2180-06-27 18:49:00, 2180-0~
$ deathtime            <dttm> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
$ admission_type       <chr> "URGENT", "EW EMER.", "EW EMER.", "EW EMER.", "EU~
$ admit_provider_id    <chr> "P49AFC", "P784FA", "P19UTS", "P06OTX", "P39NWO",~
$ admission_location   <chr> "TRANSFER FROM HOSPITAL", "EMERGENCY ROOM", "EMER~
$ discharge_location   <chr> "HOME", "HOME", "HOSPICE", "HOME", NA, "HOME HEAL~
$ insurance            <chr> "Medicaid", "Medicaid", "Medicaid", "Medicaid", N~
$ language             <chr> "English", "English", "English", "English", "Engl~
$ marital_status       <chr> "WIDOWED", "WIDOWED", "WIDOWED", "WIDOWED", "SING~
$ race                 <chr> "WHITE", "WHITE", "WHITE", "WHITE", "WHITE", "WHI~
$ edregtime            <dttm> 2180-05-06 19:17:00, 2180-06-26 15:54:00, 2180-0~
```

```
$ edouttime            <dttm> 2180-05-06 23:30:00, 2180-06-26 21:31:00, 2180-0~
$ hospital_expire_flag <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
```

```
# `fread` (data.table) speed, memory, and data types
fread <- system.time({
  df_fread <- fread(path)
})
mem_fread <- pryr::object_size(df_fread)
glimpse(df_fread)
```

```
Rows: 546,028
Columns: 16
$ subject_id           <int> 10000032, 10000032, 10000032, 10000032, 10000068,~
$ hadm_id              <int> 22595853, 22841357, 25742920, 29079034, 25022803,~
$ admittime            <dttm> 2180-05-06 22:23:00, 2180-06-26 18:27:00, 2180-0~
$ dischtime            <dttm> 2180-05-07 17:15:00, 2180-06-27 18:49:00, 2180-0~
$ deathtime            <dttm> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
$ admission_type       <chr> "URGENT", "EW EMER.", "EW EMER.", "EW EMER.", "EU~
$ admit_provider_id    <chr> "P49AFC", "P784FA", "P19UTS", "P06OTX", "P39NWO",~
$ admission_location   <chr> "TRANSFER FROM HOSPITAL", "EMERGENCY ROOM", "EMER~
$ discharge_location   <chr> "HOME", "HOME", "HOSPICE", "HOME", "", "HOME HEAL~
$ insurance            <chr> "Medicaid", "Medicaid", "Medicaid", "Medicaid", "~
$ language             <chr> "English", "English", "English", "English", "Engl~
$ marital_status       <chr> "WIDOWED", "WIDOWED", "WIDOWED", "WIDOWED", "SING~
$ race                 <chr> "WHITE", "WHITE", "WHITE", "WHITE", "WHITE", "WHI~
$ edregtime            <dttm> 2180-05-06 19:17:00, 2180-06-26 15:54:00, 2180-0~
$ edouttime            <dttm> 2180-05-06 23:30:00, 2180-06-26 21:31:00, 2180-0~
$ hospital_expire_flag <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
```

```
result <- data.frame(
  read_types = c("Base R (read.csv)",
                 "Tidyverse (read_csv)",
                 "Data.table (fread)"),
  speed_sec = c(baseR[3], tidyverse[3], fread[3]),
  memory_MB = c(as.numeric(mem_base)/1e6,
                as.numeric(mem_tidyverse)/1e6,
                as.numeric(mem_fread)/1e6)
)
```

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.)

The fastest function for reading the file was fread from the data.table package, with a reading time of 0.663 seconds.

Yes, there are differences in the parsed data types: Base R (read.csv): Converts character columns to factors by default. Tidyverse (read_csv): Keeps character columns as characters. Data.table (fread): Also keeps character columns as characters. So, the main difference is that Base R converts character columns to factors, while both Tidyverse and Data.table preserve them as characters.

The most memory-efficient function was fread from data.table, which used 63.47 MB of memory. read_csv from tidyverse also performed better than base R in terms of memory usage, consuming 70.02 MB. read.csv from base R was the most memory-intensive, using 200.10 MB of memory.

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`.)

```
# Check summary of the deathtime column
summary(df_tidyverse$deathtime)
```

```
                     Min.                      1st Qu.
"2110-01-25 09:40:00.0000" "2133-10-02 13:43:15.0000"
                   Median                         Mean
"2153-11-01 23:12:30.0000" "2153-10-10 11:43:57.6030"
                  3rd Qu.                         Max.
"2173-10-06 11:06:45.0000" "2214-10-12 12:51:00.0000"
                     NA's
                 "534238"
```

```
# Check number of missing values (NA)
sum(is.na(df_tidyverse$deathtime))
```

```
[1] 534238
```

```
col_types_defined <- cols(
  subject_id = col_double(),
  hadm_id = col_double(),
  admittime = col_datetime(),
  dischtime = col_datetime(),
  deathtime = col_datetime(),
  admission_type = col_character(),
  admit_provider_id = col_character(),
  admission_location = col_character(),
  discharge_location = col_character(),
  insurance = col_character(),
  language = col_character(),
  marital_status = col_character(),
  race = col_character(),
  edregtime = col_datetime(),
  edouttime = col_datetime(),
  hospital_expire_flag = col_double()
)

mem_time_reingest <- system.time({
  df_reingest <- read_csv("~/mimic/hosp/admissions.csv.gz",
                          col_types = col_types_defined)
})

cat("Default read_csv time:", tidyverse[3], "seconds\n")
```

```
Default read_csv time: 0.675 seconds
```

```
cat("Default read_csv memory:", as.numeric(mem_tidyverse)/1e6, "MB\n")
```

```
Default read_csv memory: 70.02259 MB
```

By re-ingesting the admissions.csv.gz file and specifying column data types using the col_types argument in read_csv, the run time does not significantly change compared to the default ingestion method:

Run Time: The re-ingestion time using the specified column types is 1.021 seconds, which is the same as the default read time.

Memory Usage: The memory usage after specifying column types is 70.02 MB, which is similar to the default method.

Thus, explicitly specifying the column data types does not notably change the speed or memory usage in this case. However, defining the data types can help optimize performance when dealing with large datasets.

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 justina  staff  2592909134 Oct  3 06:08 /Users/justina/mimic/hosp/labevents.csv
```

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,"
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,"C
```

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.

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

labevents_time <- system.time({
  df_labevents <- read_csv("~/mimic/hosp/labevents.csv.gz")
})
```

When trying to read labevents.csv.gz using read_csv, I encountered the following issues: Long reading time. After running for about 3 minutes, the reading process still hadn't completed, forcing me to manually abort the program. The file is very large, and read_csv is not able to efficiently handle such a large dataset, resulting in a performance bottleneck.

Memory consumption: Due to the file's substantial size, read_csv requires a significant amount of memory to load the entire dataset, which could exhaust the available memory on the computer.

Thus, read_csv is suitable for smaller datasets, but for large files like labevents.csv.gz, its performance is inadequate, leading to extremely slow read times or failure to complete the operation.

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_selected <- read_csv(
  path_labevents,
  col_select = c(subject_id, itemid, charttime, valuenum)
)

print(labevents_selected, width = Inf)
```

By selecting only the necessary columns, I successfully overcame the performance issues related to large files. Using read_csv with the col_select argument significantly improved the speed and made memory consumption more manageable.

For large datasets like this one, selectively reading the required columns is an effective strategy that reduces the data load, thus enhancing performance and preventing memory overflow.

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`?

```
zcat < ~/mimic/hosp/labevents.csv.gz | awk -F',' '{if ($5 == 50912 || \
$5 == 50971 || $5 == 50983 || $5 == 50902 || $5 == 50882 || $5 == 51221 || \
$5 == 51301 || $5 == 50931) print $2 "," $5 "," $7 "," $10}' | \
gzip > ~/203b/hw/hw2/labevents_filtered.csv.gz
```

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

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

```
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
 32679895
```

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

```
Rows: 32679895 Columns: 4
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl  (3): 10000032, 50931, 95
dttm (1): 2180-03-23 11:51:00

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
 17.676   1.464   7.181
```

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 > ~/203b/hw/hw2/labevents.csv
```

Then use `arrow::open_dataset` to ingest `labevents.csv`, select columns, and filter `itemid` as in Q2.3. How long does the ingest+select+filter process take? Display the number of rows and the first 10 rows of the result tibble, and make sure they match those in Q2.3. (Hint: use `dplyr` verbs for selecting columns and filtering rows.)

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

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

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

```
   user  system elapsed
 44.084   3.480  41.505
```

```
nrow(labevents_filtered_arrow)
```

```
[1] 32679896
```

```
head(labevents_filtered_arrow, 10)
```

```
# A tibble: 10 x 4
   subject_id itemid charttime           valuenum
        <int>  <int> <dttm>                 <dbl>
```

```
 1   10000032  50931 2180-03-23 04:51:00     95
 2   10000032  50882 2180-03-23 04:51:00     27
 3   10000032  50902 2180-03-23 04:51:00    101
 4   10000032  50912 2180-03-23 04:51:00      0.4
 5   10000032  50971 2180-03-23 04:51:00      3.7
 6   10000032  50983 2180-03-23 04:51:00    136
 7   10000032  51221 2180-03-23 04:51:00     45.4
 8   10000032  51301 2180-03-23 04:51:00      3
 9   10000032  51221 2180-05-06 15:25:00     42.6
10   10000032  51301 2180-05-06 15:25:00      5
```

Apache Arrow is designed to handle huge datasets that do not fit into memory, a situation very common in fields like data science and machine learning. Arrow provides an extremely optimized columnar memory format for efficiently doing analytics with no repeated reading/writing of data. In addition, it has great applications in distributed computing, for example, big data, where you need to move or process huge datasets in record time.

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.

```
# Write dataset from CSV to Parquet format
write_dataset(open_dataset("labevents.csv", format = "csv"),
              path = "labevents_parquet", format = "parquet")
```

```
# Check the size of the Parquet file
system("du -sh labevents_parquet")

# Timing the ingestion, filtering, and processing of the Parquet file
system.time({
  labevents_parquet <- open_dataset("labevents_parquet", format = "parquet") %>%
    filter(itemid %in% c(50912, 50971, 50983, 50902,
                         50882, 51221, 51301, 50931)) %>%
    select(subject_id, itemid, charttime, valuenum) %>%
    collect() %>%  # Collect the data into R
```

```
    # Ensure charttime is in UTC to prevent timezone mismatch
    mutate(charttime = as.POSIXct(charttime, format="%Y-%m-%d %H:%M:%S",
                                  tz="UTC")) %>%
    arrange(subject_id, charttime)  # Arrange by subject_id and charttime
})
```

```
   user  system elapsed
 12.538   3.167   6.260
```

```
# Print the number of rows in the filtered data
cat("Number of rows:", nrow(labevents_parquet), "\n")
```

```
Number of rows: 32679896
```

```
# Display the first 10 rows of the result
print(head(labevents_parquet, 10))
```

```
# A tibble: 10 x 4
   subject_id itemid charttime           valuenum
        <int>  <int> <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
```

Parquet is a columnar storage format that is ideal for big data processing. Let's assume you have a huge spreadsheet comprising millions of rows and thousands of columns, and you want to analyze just a couple of the columns-with the above information in mind, the traditional row-based information would require you to bring the data into memory all at once, whereas you actually need a small part. Parquet achieves this through its columnar file format, where you only have to load those columns you are interested in. This makes it much more efficient with respect to both storage and query speed. That finds broad applications for big data analytics, especially when systems like Apache Hadoop or Spark come into the picture.

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.

```
execution_time <- system.time({
  con <- dbConnect(duckdb::duckdb())

  labevents_arrow <- arrow::open_dataset("labevents_parquet",
                                         format = "parquet")

  labevents_duckdb <- arrow::to_duckdb(labevents_arrow, con = con, table_name = "labevents")

  filtered_data <- tbl(con, "labevents") %>%
    filter(itemid %in% c(50912, 50971, 50983, 50902, 50882,
                         51221, 51301, 50931)) %>%
    select(subject_id, itemid, charttime, valuenum) %>%
    collect() %>%  # Collect data into R for processing
    mutate(charttime = as.POSIXct(charttime,
                                  format="%Y-%m-%d %H:%M:%S", tz="UTC")) %>%
    # Convert charttime to UTC
    arrange(subject_id, charttime)

  # Arrange the data by subject_id and charttime
  dbDisconnect(con, shutdown = TRUE)
})

print(execution_time)
```

```
   user  system elapsed
 13.447   3.070   5.783
```

```
cat("Number of rows:", nrow(filtered_data), "\n")
```

```
Number of rows: 32679896
```

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

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

DuckDB is an in-memory, column-oriented, analytical database. It's a tiny database that resides directly in your program or in your computer, not on any server, and does not require any installation. Think of it as a fast, local, in-memory database that is perfect for quick processing of volumes of data. DuckDB is able to execute SQL queries on datasets an order of magnitude faster than traditional systems and is especially optimized for analytical tasks such as filtering, aggregation, and joining data. It is easy to set up, use, and its strong performance when working with big data directly in memory; that is why it is perfect for data analysis on smaller and medium-sized 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,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
```

```
 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
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.

```
zcat < ~/mimic/icu/chartevents.csv.gz | \
awk -F',' '{if ($7 == 220045 || $7 == 220181 || $7 == 220179 || \
$7 == 223761 || $7 == 220210) print $7}' | \
gzip > ~/203b/hw/hw2/chartevents_filtered.csv
```

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

```
zcat < chartevents_filtered.csv | wc -l
```

```
 30195426
```

```
zcat < chartevents_filtered.csv | head -10
```

```
223761
220179
220181
220045
220210
220045
220179
220181
220210
220045
```