Biostat 203B Homework 2

Due Feb 7, 2025 @ 11:59PM

Palash Raval and 406551574

Display machine information for reproducibility:

```
sessionInfo()
```

```
R version 4.3.1 (2023-06-16)
Platform: aarch64-apple-darwin20 (64-bit)
Running under: macOS Sonoma 14.2.1

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.3-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_4.3.1    fastmap_1.2.0     cli_3.6.3         tools_4.3.1
 [5] htmltools_0.5.8.1 rstudioapi_0.17.1 yaml_2.3.10       rmarkdown_2.28
 [9] knitr_1.48        jsonlite_1.8.9    xfun_0.48         digest_0.6.37
[13] rlang_1.1.4       evaluate_1.0.1
```

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

```
library(arrow)
```

```
Warning: package 'arrow' was built under R version 4.3.3
```

```

Attaching package: 'arrow'

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

    timestamp
```

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

```
Warning: package 'data.table' was built under R version 4.3.3
```

```
library(duckdb)
```

```
Warning: package 'duckdb' was built under R version 4.3.3

Loading required package: DBI

Warning: package 'DBI' was built under R version 4.3.3
```

```
library(memuse)
```

```
Warning: package 'memuse' was built under R version 4.3.3
```

```
library(pryr)
```

```

Attaching package: 'pryr'

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

    address
```

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

```
Loading required package: R.oo

Warning: package 'R.oo' was built under R version 4.3.3

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, warnings
```

```
library(tidyverse)
```

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

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

Display memory information of your computer

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

```
Totalram:  16.000 GiB
Freeram:    4.522 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 0
-rw-r--r--@ 1 palashraval  staff    19928140 Jun 24  2024 admissions.csv.gz
-rw-r--r--@ 1 palashraval  staff      427554 Apr 12  2024 d_hcpcs.csv.gz
-rw-r--r--@ 1 palashraval  staff      876360 Apr 12  2024 d_icd_diagnoses.csv.gz
-rw-r--r--@ 1 palashraval  staff      589186 Apr 12  2024 d_icd_procedures.csv.gz
-rw-r--r--@ 1 palashraval  staff       13169 Oct  3 09:07 d_labitems.csv.gz
-rw-r--r--@ 1 palashraval  staff    33564802 Oct  3 09:07 diagnoses_icd.csv.gz
-rw-r--r--@ 1 palashraval  staff     9743908 Oct  3 09:07 drgcodes.csv.gz
-rw-r--r--@ 1 palashraval  staff   811305629 Apr 12  2024 emar.csv.gz
-rw-r--r--@ 1 palashraval  staff   748158322 Apr 12  2024 emar_detail.csv.gz
-rw-r--r--@ 1 palashraval  staff     2162335 Apr 12  2024 hcpcsevents.csv.gz
-rw-r--r--@ 1 palashraval  staff  2592909134 Oct  3 09:08 labevents.csv.gz
-rw-r--r--@ 1 palashraval  staff   117644075 Oct  3 09:08 microbiologyevents.csv.gz
-rw-r--r--@ 1 palashraval  staff    44069351 Oct  3 09:08 omr.csv.gz
-rw-r--r--@ 1 palashraval  staff     2835586 Apr 12  2024 patients.csv.gz
-rw-r--r--@ 1 palashraval  staff   525708076 Apr 12  2024 pharmacy.csv.gz
-rw-r--r--@ 1 palashraval  staff   666594177 Apr 12  2024 poe.csv.gz
-rw-r--r--@ 1 palashraval  staff    55267894 Apr 12  2024 poe_detail.csv.gz
-rw-r--r--@ 1 palashraval  staff   606298611 Apr 12  2024 prescriptions.csv.gz
-rw-r--r--@ 1 palashraval  staff     7777324 Apr 12  2024 procedures_icd.csv.gz
-rw-r--r--@ 1 palashraval  staff      127330 Apr 12  2024 provider.csv.gz
-rw-r--r--@ 1 palashraval  staff     8569241 Apr 12  2024 services.csv.gz
-rw-r--r--@ 1 palashraval  staff    46185771 Oct  3 09:08 transfers.csv.gz
```

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

```
total 6840736
-rw-r--r--@ 1 palashraval  staff       41566 Apr 12  2024 caregiver.csv.gz
-rw-r--r--@ 1 palashraval  staff  3502392765 Apr 12  2024 chartevents.csv.gz
-rw-r--r--@ 1 palashraval  staff       58741 Apr 12  2024 d_items.csv.gz
-rw-r--r--@ 1 palashraval  staff    63481196 Apr 12  2024 datetimeevents.csv.gz
-rw-r--r--@ 1 palashraval  staff     3342355 Oct  3 07:36 icustays.csv.gz
-rw-r--r--@ 1 palashraval  staff   311642048 Apr 12  2024 ingredientevents.csv.gz
-rw-r--r--@ 1 palashraval  staff   401088206 Apr 12  2024 inputevents.csv.gz
-rw-r--r--@ 1 palashraval  staff    49307639 Apr 12  2024 outputevents.csv.gz
-rw-r--r--@ 1 palashraval  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.)

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

```
   user  system elapsed
  5.673   0.070  48.463
```

```
pryr::object_size(read.csv("~/mimic/hosp/admissions.csv.gz"))
```

```
200.10 MB
```

```
system.time(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
  1.059   0.132   0.652
```

```
pryr::object_size(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.
```

```
70.02 MB
```

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

```
   user  system elapsed
  0.473   0.168   0.334
```

```
pryr::object_size(fread("~/mimic/hosp/admissions.csv.gz"))
```

```
63.47 MB
```

Solution: fread() is the function with the fastest speed(~ 0.548 seconds). read_csv() is second with a speed of ~1.103 seconds and read.csv() being the slowest with a speed of ~5.758 seconds. Yes, there is a difference in the default parsed data types: read.csv() provides a dataframe, read_csv() gives a tibble, and fread() returns a data.table. fread() is the most memory efficient with a usage of ~63.47 MB, with read_csv() coming in second with ~70.02 MB, and read.csv() using the most memory with ~200.10 MB.

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`.)

```
system.time(read_csv("~/mimic/hosp/admissions.csv.gz",
                     col_types = cols(subject_id = "i",
                                      hadm_id = "i",admittime = "T",
                                      dischtime = "T",deathtime = "T",
                                      admission_type = "c",
                                      admit_provider_id = "c",
                                      admission_location = "c",
                                      discharge_location = "c",
                                      insurance = "c", language = "c",
                                      marital_status = "c",
                                      race = "c", edregtime = "T",
                                      edouttime = "T",
                                      hospital_expire_flag = "i")))
```

```
   user  system elapsed
  0.998   0.118   0.568
```

```
pryr::object_size(read_csv("~/mimic/hosp/admissions.csv.gz",
                           col_types = cols(subject_id = "i",
                                            hadm_id = "i",admittime = "T",
                                            dischtime = "T",deathtime = "T",
                                            admission_type = "c",
                                            admit_provider_id = "c",
                                            admission_location = "c",
                                            discharge_location = "c",
                                            insurance = "c", language = "c",
                                            marital_status = "c",
                                            race = "c", edregtime = "T",
                                            edouttime = "T",
                                            hospital_expire_flag = "i")))
```

```
63.47 MB
```

Solution: The speed of read_csv() slightly decreased after I indicated the appropriate data types for each of the columns. It went from an original speed of

1.103 seconds, to a speed of 1.073 seconds. The memory usage also seemed to decreased after I indicated the appropriate data types for each column. Originally, the memory usage was 70.02 MB and for this question it was shown to be 63.47 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 palashraval  staff  2592909134 Oct  3 09:08 /Users/palashraval/mimic/hosp/labev
```

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,"(
```

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.

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

Solution: It was running for more than three minutes on my computer, so I aborted the program. It is supposed to crash R Studio because the data file is so large than you cannot just run it without some amount of modification/alteration.

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`.)

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

Solution: No, it did not resolve the ingestion issue. Even though I ingested only those four columns, R Studio still crashed while I ran the modified command.

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 ',' '$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
```

```
zcat: can't read stdin: Operation canceled
```

```
zcat < labevents_filtered.csv.gz | wc -l
```

```
zcat: can't read stdin: Operation canceled
       0
```

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

Solution: There are 32679896 lines in the filtered data file `labevents_filtered.csv.gz`. It took about 50 seconds for the read_csv() function to ingest this filtered data file.

Q2.4 Ingest `labevents.csv` by Apache Arrow

Our second strategy is to use Apache Arrow for larger-than-memory data analytics. Unfortunately Arrow does not work with gz files directly. First decompress `labevents.csv.gz` to `labevents.csv` and put it in the current working directory (do not add it in git!). To save render time, put `#| eval: false` at the beginning of this code chunk. TA will change it to `#| eval: true` when rendering your qmd file.

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

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

```
gzip -dc < ~/mimic/hosp/labevents.csv.gz > ./labevents.csv
```

```
arrow_filtered = arrow::open_dataset("./labevents.csv", format = "csv") %>%
  select(subject_id, itemid, charttime, valuenum) %>%
  filter(itemid %in% c("50912","50971", "50983", "50902",
                       "50882","51221", "51301", "50931")) %>%
  collect()
```

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

```
nrow(arrow_filtered)
```

```
[1] 32679896
```

```
head(arrow_filtered, 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
```

Solution: It took about 45 seconds for it to run. There are 32679896 rows in this filtered data, which matches with the number of rows that I found in the previous question. Apache Arrow is a library that can process and move massive data sets. It can even move data sets from different languages, such as Python. It essentially organizes data by columns instead of by rows, in a process that is called "in-memory columnar format". "In-memory" means it runs locally and not on some server. It is the reason why Apache Arrow is faster and more efficient for dealing with large data sets that would otherwise by a hastle to deal with.

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.

```
arrow_dataset = arrow::open_dataset("./labevents.csv", format = "csv")

arrow::write_dataset(arrow_dataset, "./labevents_parquet", format = "parquet")
```

```
ls -lh ./labevents_parquet
```

```
total 5341192
-rw-r--r--@ 1 palashraval  staff   2.5G Feb  6 19:04 part-0.parquet
```

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

```
parquet_filtered = arrow::open_dataset("./labevents_parquet", format = "parquet") %>%
  select(subject_id, itemid, charttime, valuenum) %>%
  filter(itemid %in% c(50912,50971, 50983, 50902,
                       50882,51221, 51301, 50931)) %>%
  collect()
```

```
nrow(parquet_filtered)
```

```
[1] 32679896
```

```
head(parquet_filtered, 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
```

Solution: The parquet data file is 2.5G. It took about 7 seconds for the ingest+select+filter process. The number of rows and the first 10 rows match what was found in Q2.3. Parquet format is a data file format that is column-oriented. It is helpful for large data sets because it has a more efficient system for storage, so filtering and retrieving the data will be much faster than if you were to stick with the csv format. This was proven by how fast the ingest+select+filter process occurred compared to the methods done before.

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.

```
duck_parquet = open_dataset("./labevents_parquet") %>%
  to_duckdb() %>%
  select(subject_id, itemid, charttime, valuenum) %>%
  filter(itemid %in% c(50912,50971, 50983, 50902, 50882, 51221, 51301,
                       50931)) %>%
  collect()
```

```
system.time(open_dataset("./labevents_parquet") %>%
  to_duckdb() %>%
  select(subject_id, itemid, charttime, valuenum) %>%
  filter(itemid %in% c(50912,50971, 50983, 50902, 50882, 51221, 51301,
                       50931)) %>%
  collect())
```

```
nrow(duck_parquet)
```

```
[1] 32679896
```

```
head(duck_parquet, 10)
```

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

```
        <dbl>  <dbl> <dttm>                 <dbl>
 1   10002155  50971 2129-08-15 06:25:00     5.4
 2   10002155  50983 2129-08-15 06:25:00   128
 3   10002155  51221 2129-08-15 06:25:00    29.4
 4   10002155  51301 2129-08-15 06:25:00     5.8
 5   10002155  50882 2129-08-15 15:20:00    25
 6   10002155  50902 2129-08-15 15:20:00    95
 7   10002155  50912 2129-08-15 15:20:00     1.1
 8   10002155  50931 2129-08-15 15:20:00    98
 9   10002155  50971 2129-08-15 15:20:00     5
10   10002155  50983 2129-08-15 15:20:00   128
```

```
head(duck_parquet %>% arrange(subject_id), 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
```

Solution: It took about 6.716 seconds for the ingest+select+filter process. The number of rows was 32679896, which is the same as the number of rows for Q2.3. When I did head(), I got different values than what I did previously. It was only after I arranged in ascending order did the first 10 rows match the ones from Q2.3. DuckDB is a database that can be utilized in R through the Arrow Apache library. It can also be used in other programming languages as well, which makes it versatile. DuckDB makes querying quick and efficient because it is column-focused(columnar), which means it stores data by columns, meaning that less storage is taken up and the processing will be faster. This makes it a great option when you have to deal with data analysis 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,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 million.

```
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.

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

```
zcat < ~/mimic/icu/chartevents.csv.gz |
  awk -F ',' '$7 == "220045" || $7 == "220181" || $7 == "220179" ||
  $7 == "223761" || $7 == "220210" {print $0}' |
  gzip > chartevents_filtered.csv.gz
```

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

```
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 14:17:00,220181,56,56,mmHg,0
10000032,29079034,39553978,18704,2180-07-23 14:12:00,2180-07-23 14:17:00,220045,91,91,bpm,0
10000032,29079034,39553978,18704,2180-07-23 14:12:00,2180-07-23 14:17:00,220210,24,24,insp/m
10000032,29079034,39553978,18704,2180-07-23 14:30:00,2180-07-23 14:43:00,220045,93,93,bpm,0
10000032,29079034,39553978,18704,2180-07-23 14:30:00,2180-07-23 14:43:00,220179,95,95,mmHg,0
10000032,29079034,39553978,18704,2180-07-23 14:30:00,2180-07-23 14:43:00,220181,67,67,mmHg,0
10000032,29079034,39553978,18704,2180-07-23 14:30:00,2180-07-23 14:43:00,220210,21,21,insp/m
10000032,29079034,39553978,18704,2180-07-23 15:00:00,2180-07-23 15:34:00,220045,94,94,bpm,0
```

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

```
30195426
```

Solution: I used Bash to get a subset. I used `zcat` to deal with the compressed file and filtered based on the values in itemid, so I would be left with only the rows that had the itemid values I was interested in. After this, I used `gzip` to compress the subset file. After filtering the original 'chartevents.csv.gz' file for the specific itemid values, there are 30195426 rows in the subset of 'chartevents.csv.gz'.