Biostat 203B Homework 2

Due Jan 24, 2025 @ 11:59PM

Luke Hodges 906182810

Display machine information for reproducibility:

```
sessionInfo()
```

```
R version 4.4.2 (2024-10-31)
Platform: x86_64-apple-darwin20
Running under: macOS Sequoia 15.0

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

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

time zone: America/Los_Angeles
tzcode source: internal

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

loaded via a namespace (and not attached):
 [1] compiler_4.4.2    fastmap_1.2.0     cli_3.6.3         tools_4.4.2
 [5] htmltools_0.5.8.1 rstudioapi_0.17.1 yaml_2.3.10       rmarkdown_2.29
 [9] knitr_1.49        jsonlite_1.8.9    xfun_0.50         digest_0.6.37
[13] rlang_1.1.4       evaluate_1.0.1
```

Load necessary libraries (tidyverse, data.table, arrow, etc. Hide the messages because they keep doing it even though its rendered and put into the library)

```r
library(arrow)
library(data.table)
library(duckdb)
library(memuse)
library(pryr)
library(R.utils)
library(tidyverse)
library(dplyr)
```

Display memory information of your computer

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

```
Totalram:  32.000 GiB
Freeram:   22.090 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:

```bash
ls -l ~/mimic/hosp/
```

```
total 48888024
-rw-r--r--@ 1 lukehodges  staff     19928140 Jun 24  2024 admissions.csv.gz
-rw-r--r--@ 1 lukehodges  staff       427554 Apr 12  2024 d_hcpcs.csv.gz
-rw-r--r--@ 1 lukehodges  staff       876360 Apr 12  2024 d_icd_diagnoses.csv.gz
-rw-r--r--@ 1 lukehodges  staff       589186 Apr 12  2024 d_icd_procedures.csv.gz
-rw-r--r--@ 1 lukehodges  staff        13169 Oct  3 09:07 d_labitems.csv.gz
-rw-r--r--@ 1 lukehodges  staff     33564802 Oct  3 09:07 diagnoses_icd.csv.gz
-rw-r--r--@ 1 lukehodges  staff      9743908 Oct  3 09:07 drgcodes.csv.gz
-rw-r--r--@ 1 lukehodges  staff    811305629 Apr 12  2024 emar.csv.gz
-rw-r--r--@ 1 lukehodges  staff    748158322 Apr 12  2024 emar_detail.csv.gz
-rw-r--r--@ 1 lukehodges  staff      2162335 Apr 12  2024 hcpcsevents.csv.gz
drwxr-xr-x@ 3 lukehodges  staff           96 Jan 30 16:17 labevents
-rw-r--r--@ 1 lukehodges  staff  18402851720 Jan 20 22:00 labevents.csv
-rw-r--r--@ 1 lukehodges  staff   2592909134 Oct  3 09:08 labevents.csv.gz
drwxr-xr-x@ 3 lukehodges  staff           96 Jan 30 22:13 labevents.filtered
-rw-r--r--@ 1 lukehodges  staff    174094381 Jan 30 21:01 labevents_filtered.csv.gz
-rw-r--r--@ 1 lukehodges  staff    117644075 Oct  3 09:08 microbiologyevents.csv.gz
-rw-r--r--@ 1 lukehodges  staff     44069351 Oct  3 09:08 omr.csv.gz
```

```
-rw-r--r--@ 1 lukehodges  staff    152917918 Jan 30 16:08 part-0.parquet
-rw-r--r--@ 1 lukehodges  staff      2835586 Apr 12  2024 patients.csv.gz
-rw-r--r--@ 1 lukehodges  staff    525708076 Apr 12  2024 pharmacy.csv.gz
-rw-r--r--@ 1 lukehodges  staff    666594177 Apr 12  2024 poe.csv.gz
-rw-r--r--@ 1 lukehodges  staff     55267894 Apr 12  2024 poe_detail.csv.gz
-rw-r--r--@ 1 lukehodges  staff    606298611 Apr 12  2024 prescriptions.csv.gz
-rw-r--r--@ 1 lukehodges  staff      7777324 Apr 12  2024 procedures_icd.csv.gz
-rw-r--r--@ 1 lukehodges  staff       127330 Apr 12  2024 provider.csv.gz
-rw-r--r--@ 1 lukehodges  staff      8569241 Apr 12  2024 services.csv.gz
-rw-r--r--@ 1 lukehodges  staff     46185771 Oct  3 09:08 transfers.csv.gz
```

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

```
total 90426912
-rw-r--r--@ 1 lukehodges  staff        41566 Apr 12  2024 caregiver.csv.gz
-rw-r--r--@ 1 lukehodges  staff  41935806083 Feb  7 15:57 chartevents.csv
-rw-r--r--@ 1 lukehodges  staff   3502392765 Apr 12  2024 chartevents.csv.gz
-rw-r--r--@ 1 lukehodges  staff        58741 Apr 12  2024 d_items.csv.gz
-rw-r--r--@ 1 lukehodges  staff     63481196 Apr 12  2024 datetimeevents.csv.gz
-rw-r--r--@ 1 lukehodges  staff      3342355 Oct  3 07:36 icustays.csv.gz
-rw-r--r--@ 1 lukehodges  staff    311642048 Apr 12  2024 ingredientevents.csv.gz
-rw-r--r--@ 1 lukehodges  staff    401088206 Apr 12  2024 inputevents.csv.gz
-rw-r--r--@ 1 lukehodges  staff     49307639 Apr 12  2024 outputevents.csv.gz
-rw-r--r--@ 1 lukehodges  staff     24096834 Apr 12  2024 procedureevents.csv.gz
```

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.

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.

The system.time for the base R command is:

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

```
   user  system elapsed 
 10.090   0.250  10.348 
```

Here, the user time is 11.046, the system is 0.222, and the elapsed time is 11.331

The system.time for by using the read_csv command in tidyverse is

```r
system.time({
  admissions.tidy <- 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 
  3.638   0.648   1.709 
```

Here, the user time is 3.915, the system is 0.391, and the elapsed time is 2.378.

Compared to the R one, the user and elapsed time is faster, but the system is slower

The system.time for the fread in the data.table package

```r
system.time({
  admissions.data <- fread("~/mimic/hosp/admissions.csv.gz")
})
```

```
   user  system elapsed 
  2.991   0.166   0.853 
```

Here, the user, system, and elapsed time is 2.87, 0.19, and 0.984, respectively

Which function is fastest?

Answer: The function that is the fastest is the fread in the data.table package. The data.table package is the fastest, followed by the tidyverse, and the base R command is the slowest for the user. For the system, the data.table package is the fastest, followed by the base R command, and the tidyverse is the slowest. For the elapsed time, the data.table package is the fastest, followed by the tidyverse, and the base R command is the slowest

Is there difference in the (default) parsed data types?

To look at the structure and default parsed data types, we use the str command

```
str(admissions.r)
```

```
'data.frame':   546028 obs. of  16 variables:
 $ subject_id          : int  10000032 10000032 10000032 10000032 10000068 10000084 10000084
 $ hadm_id             : int  22595853 22841357 25742920 29079034 25022803 23052089 29888819
 $ admittime           : chr  "2180-05-06 22:23:00" "2180-06-26 18:27:00" "2180-08-05 23:44:0
 $ dischtime           : chr  "2180-05-07 17:15:00" "2180-06-27 18:49:00" "2180-08-07 17:50:0
 $ deathtime           : chr  "" "" "" "" ...
 $ admission_type      : chr  "URGENT" "EW EMER." "EW EMER." "EW EMER." ...
 $ admit_provider_id   : chr  "P49AFC" "P784FA" "P19UTS" "P06OTX" ...
 $ admission_location  : chr  "TRANSFER FROM HOSPITAL" "EMERGENCY ROOM" "EMERGENCY ROOM" "EME
 $ discharge_location  : chr  "HOME" "HOME" "HOSPICE" "HOME" ...
 $ insurance           : chr  "Medicaid" "Medicaid" "Medicaid" "Medicaid" ...
 $ language            : chr  "English" "English" "English" "English" ...
 $ marital_status      : chr  "WIDOWED" "WIDOWED" "WIDOWED" "WIDOWED" ...
 $ race                : chr  "WHITE" "WHITE" "WHITE" "WHITE" ...
 $ edregtime           : chr  "2180-05-06 19:17:00" "2180-06-26 15:54:00" "2180-08-05 20:58:0
 $ edouttime           : chr  "2180-05-06 23:30:00" "2180-06-26 21:31:00" "2180-08-06 01:44:0
 $ hospital_expire_flag: int  0 0 0 0 0 0 0 0 0 0 ...
```

```
str(admissions.tidy)
```

```
spc_tbl_ [546,028 x 16] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
 $ subject_id          : num [1:546028] 1e+07 1e+07 1e+07 1e+07 1e+07 ...
 $ hadm_id             : num [1:546028] 22595853 22841357 25742920 29079034 25022803 ...
 $ admittime           : POSIXct[1:546028], format: "2180-05-06 22:23:00" "2180-06-26 18:27:0
 $ dischtime           : POSIXct[1:546028], format: "2180-05-07 17:15:00" "2180-06-27 18:49:0
 $ deathtime           : POSIXct[1:546028], format: NA NA ...
 $ admission_type      : chr [1:546028] "URGENT" "EW EMER." "EW EMER." "EW EMER." ...
 $ admit_provider_id   : chr [1:546028] "P49AFC" "P784FA" "P19UTS" "P06OTX" ...
 $ admission_location  : chr [1:546028] "TRANSFER FROM HOSPITAL" "EMERGENCY ROOM" "EMERGENCY
```

```
 $ discharge_location  : chr [1:546028] "HOME" "HOME" "HOSPICE" "HOME" ...
 $ insurance           : chr [1:546028] "Medicaid" "Medicaid" "Medicaid" "Medicaid" ...
 $ language            : chr [1:546028] "English" "English" "English" "English" ...
 $ marital_status      : chr [1:546028] "WIDOWED" "WIDOWED" "WIDOWED" "WIDOWED" ...
 $ race                : chr [1:546028] "WHITE" "WHITE" "WHITE" "WHITE" ...
 $ edregtime           : POSIXct[1:546028], format: "2180-05-06 19:17:00" "2180-06-26 15:54:0
 $ edouttime           : POSIXct[1:546028], format: "2180-05-06 23:30:00" "2180-06-26 21:31:0
 $ hospital_expire_flag: num [1:546028] 0 0 0 0 0 0 0 0 0 0 ...
 - attr(*, "spec")=
  .. cols(
  ..   subject_id = col_double(),
  ..   hadm_id = col_double(),
  ..   admittime = col_datetime(format = ""),
  ..   dischtime = col_datetime(format = ""),
  ..   deathtime = col_datetime(format = ""),
  ..   admission_type = col_character(),
  ..   admit_provider_id = col_character(),
  ..   admission_location = col_character(),
  ..   discharge_location = col_character(),
  ..   insurance = col_character(),
  ..   language = col_character(),
  ..   marital_status = col_character(),
  ..   race = col_character(),
  ..   edregtime = col_datetime(format = ""),
  ..   edouttime = col_datetime(format = ""),
  ..   hospital_expire_flag = col_double()
  .. )
 - attr(*, "problems")=<externalptr>
```

```
str(admissions.data)
```

```
Classes 'data.table' and 'data.frame':  546028 obs. of  16 variables:
 $ subject_id          : int  10000032 10000032 10000032 10000032 10000068 10000084 10000084
 $ hadm_id             : int  22595853 22841357 25742920 29079034 25022803 23052089 29888819
 $ admittime           : POSIXct, format: "2180-05-06 22:23:00" "2180-06-26 18:27:00" ...
 $ dischtime           : POSIXct, format: "2180-05-07 17:15:00" "2180-06-27 18:49:00" ...
 $ deathtime           : POSIXct, format: NA NA ...
 $ admission_type      : chr  "URGENT" "EW EMER." "EW EMER." "EW EMER." ...
 $ admit_provider_id   : chr  "P49AFC" "P784FA" "P19UTS" "P06OTX" ...
 $ admission_location  : chr  "TRANSFER FROM HOSPITAL" "EMERGENCY ROOM" "EMERGENCY ROOM" "EMI
 $ 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>
```

The difference in the default parsed data types is that the base R command uses a data frame, the tidyverse uses a tibble, and the data.table package uses a data.table. For example, using the base r command, the first four columns are listed as: int, int, chr, and chr. Using tidyverse, they are num, num, POSIXct, and POSIXct. Using the data.table package, they are int, int, POSIXct, and POSIXct

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 memory usage for the R command

```r
pryr::object_size(admissions.r)
```

```
200.10 MB
```

It is 200.10 for the memory usage for the R command

The memory usage for the tidyverse command

```r
pryr::object_size(admissions.tidy)
```

```
70.02 MB
```

It is 70.02 MB for the memory usage for the tidyverse command for the data admissions.tidy

The memory usage for the data.table command

```r
pryr::object_size(admissions.data)
```

```
63.47 MB
```

It is 63.46 MB for the memory usage for the data.table command for the data admissions.data

The data.table package uses the least amount of memory, followed by the tidyverse, and the base R command uses the most amount 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.)

Conducting the system time by explicitly stating the column types in the read_csv command

We need to look at the colnames for the file

```
colnames(admissions.r)
```

```
 [1] "subject_id"           "hadm_id"              "admittime"
 [4] "dischtime"            "deathtime"            "admission_type"
 [7] "admit_provider_id"    "admission_location"   "discharge_location"
[10] "insurance"            "language"             "marital_status"
[13] "race"                 "edregtime"            "edouttime"
[16] "hospital_expire_flag"
```

```
system.time({
  admissions.reingest <- read_csv("~/mimic/hosp/admissions.csv.gz",
                            col_types = cols(subject_id = col_integer(),
                                             hadm_id = col_integer(),
                                             admittime = col_datetime
                                             (format = ""),
                                             dischtime = col_datetime
                                             (format = ""),
                                             deathtime = col_datetime
                                             (format = ""),
                                             admission_type =
                                               col_character(),
                                             admit_provider_id =
                                               col_character(),
                                             admission_location =
                                               col_character(),
```

```
                                                discharge_location =
                                                  col_character(),
                                                insurance = col_character(),
                                                language = col_character(),
                                                marital_status =
                                                  col_character(),
                                                race = col_character(),
                                                edregtime =
                                                  col_datetime(format = ""),
                                                edouttime =
                                                  col_datetime(format = ""),
                                                hospital_expire_flag =
                                                  col_integer()))

})
```

```
   user  system elapsed 
  3.297   0.426   1.263 
```

The run time decreases for the user, system and elapsed when using the col_types argument in read_csv. The user, system, and elapsed times are 3.340, 0.262, and 1.271, respectively. This is comparing it to the other read_csv command used to make admissions.tidy

The memory usage for the re-ingested data

```
pryr::object_size(admissions.reingest)
```

```
63.47 MB
```

The result tibble has the same memory of 63.47 MB, which is lower storage amount than the original amount of 70.02. However, this is the same as the amount from using the data.table command

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 lukehodges  staff  2592909134 Oct  3 09:08 /Users/lukehodges/mimic/hosp/labeve
```

Display the first 10 lines of this file.

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

```
labevent_id,subject_id,hadm_id,specimen_id,itemid,order_provider_id,charttime,storetime,valu
1,10000032,,2704548,50931,P69FQC,2180-03-23 11:51:00,2180-03-23 15:56:00,___,95,mg/dL,70,100
2,10000032,,36092842,51071,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,,ROUTINE,
3,10000032,,36092842,51074,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,,ROUTINE,
4,10000032,,36092842,51075,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,,ROUTINE,"
5,10000032,,36092842,51079,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,,ROUTINE,
6,10000032,,36092842,51087,P69FQC,2180-03-23 11:51:00,,,,,,,,ROUTINE,RANDOM.
7,10000032,,36092842,51089,P69FQC,2180-03-23 11:51:00,2180-03-23 16:15:00,,,,,,,ROUTINE,PRES
8,10000032,,36092842,51090,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,,ROUTINE,M
9,10000032,,36092842,51092,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,,ROUTINE,"
```

Please see above for the first ten lines of the file labevents.csv.gz

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.

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

While processing the data, I waited about ten minutes to ingest, however, I had to abort the program because it did not finish. I am assumign the data file labevents.csv.gz is a very large file (around 20 GB), which makes it harder to decompress

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.)

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

It took about 8 minutes for it to respond. The user, system, and elapsed times are: 418.660, 644.646, and 350.702, respectively. Technically, this does solve the ingestion problem after eight minutes as it was able to respond with the times compared to without the columns. With this in mind, if we were to base it off the three minute mark, then no, this did not fix the issue, but overall, yes it did

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.

Look at the first ten lines of the labevents.csv.gz file so we can understand the ordering of columns

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

```
labevent_id,subject_id,hadm_id,specimen_id,itemid,order_provider_id,charttime,storetime,valu
1,10000032,,2704548,50931,P69FQC,2180-03-23 11:51:00,2180-03-23 15:56:00,___,95,mg/dL,70,100
2,10000032,,36092842,51071,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,,ROUTINE,
3,10000032,,36092842,51074,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,,ROUTINE,
4,10000032,,36092842,51075,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,,ROUTINE,"
5,10000032,,36092842,51079,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,,ROUTINE,
6,10000032,,36092842,51087,P69FQC,2180-03-23 11:51:00,,,,,,,,ROUTINE,RANDOM.
7,10000032,,36092842,51089,P69FQC,2180-03-23 11:51:00,2180-03-23 16:15:00,,,,,,,ROUTINE,PRES
8,10000032,,36092842,51090,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,,ROUTINE,M
9,10000032,,36092842,51092,P69FQC,2180-03-23 11:51:00,2180-03-23 16:00:00,NEG,,,,,,ROUTINE,"
```

The resepctive columns are 2, 5, 7, and 10

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

```
zcat < ~/mimic/hosp/labevents.csv.gz | awk -F, '
  BEGIN {OFS=","; print "subject_id,itemid,charttime,valuenum"}
  $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
```

This file takes a while to load. While talking to Dr. Zhou, we believe that the code runs 8 different times looking specifically for each of the numbers listed above. However, even when I tried that code it still took a while to load. I am assuming this has to do with the age of my computer as it is from 2018

Checking to see how big the file size is for comparison with what is said in Slack

```
ls -lh labevents_filtered.csv.gz
```

```
-rw-r--r--@ 1 lukehodges  staff   166M Jan 30 21:24 labevents_filtered.csv.gz
```

It is 166 MB which is similar to what Dr. Zhou said in slack.

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

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

```
subject_id,itemid,charttime,valuenum
10000032,50931,2180-03-23 11:51:00,95
10000032,50882,2180-03-23 11:51:00,27
10000032,50902,2180-03-23 11:51:00,101
10000032,50912,2180-03-23 11:51:00,0.4
10000032,50971,2180-03-23 11:51:00,3.7
10000032,50983,2180-03-23 11:51:00,136
10000032,51221,2180-03-23 11:51:00,45.4
```

```
10000032,51301,2180-03-23 11:51:00,3
10000032,51221,2180-05-06 22:25:00,42.6
```

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

```
32679897
```

This is with the header. Therefore, there are actually 32679896 lines of data in the file. I used this command rather than tail -n +2 because I waited a lot longer for the other one to render

How long does it take read_csv to ingest labevents_filtered.csv.gz?

```
system.time({
  labevents.filtered.tidy <- read_csv("labevents_filtered.csv.gz")
})
```

```
Rows: 32679896 Columns: 4
-- Column specification --------------------------------------------------------
Delimiter: ","
dbl  (3): subject_id, itemid, valuenum
dttm (1): charttime

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

```
   user  system elapsed
 60.221   6.339  15.827
```

The user, system, and elapsed time is 59.259, 4.809, and 12.385, respectively. This is much faster considering that we filtered out the necessary data compared to the prior attempts

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.

First, we need to decompress labevents.csv.gz to labevents.csv to the current directory of ~/mimic

```
zcat < ~/mimic/hosp/labevents.csv.gz > labevents.csv
```

Now, let us double check that the labevents.csv file is there

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

```
-rw-r--r--@ 1 lukehodges  staff  18402851720 Jan 20 22:00 /Users/lukehodges/mimic/hosp/labeve
```

It looks like the file is successfully placed as directed

Then use arrow::open_dataset to ingest labevents.csv, select columns, and filter itemid as in Q2.3.

Using open_dataset to ingest labevents.csv. I will make an identical table to labevents.csv, but with arrow instead.

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

Now we have to select the columns and filter the new table labelevets.arrow

```
labevents.filter.arrow <- labevents.arrow %>%
  dplyr::select(subject_id, itemid, charttime, valuenum) %>%
  dplyr::filter(itemid %in% c(50912, 50971, 50983, 50902, 50882, 51221,
                              51301, 50931))
```

How long does the ingest+select+filter process take?

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

```
   user  system elapsed 
  0.038   0.010   0.040 
```

Using arrow, the user, system, and elapsed times are more faster. They are 0.047, 0.014, and 0.110, respectively

Display the number of rows.

```r
nrow(labevents.filter.arrow)
```

```
[1] 32679896
```

nrow only counts the rows of data, not the header row. Therefore, the total rows would be 32679896, which is the same as the answer above number of lines in the file labevents_filtered.csv.gz, as we saw in the previous question

First ten rows of the result tibble.

```r
first10 <- labevents.filter.arrow %>%
  head(10) %>%
  collect()
```

```r
print(first10)
```

```
# 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
```

Does this match up to the one above? Let us find out

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

```
subject_id,itemid,charttime,valuenum
10000032,50931,2180-03-23 11:51:00,95
10000032,50882,2180-03-23 11:51:00,27
10000032,50902,2180-03-23 11:51:00,101
10000032,50912,2180-03-23 11:51:00,0.4
10000032,50971,2180-03-23 11:51:00,3.7
10000032,50983,2180-03-23 11:51:00,136
10000032,51221,2180-03-23 11:51:00,45.4
10000032,51301,2180-03-23 11:51:00,3
10000032,51221,2180-05-06 22:25:00,42.6
```

Although it directly matches those in Q2.3, the only thing that does not is the charttime. The hour seems to be off, despite the minute being the same. This could be because of a time zone difference of the packages/user.

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

Apache Arrow is a development platform that allows for easier processing of data that is actively being used (in-memory), leading for quicker access and turn around. It uses a columnar format, rather than a row format which allows for easier access. It is used in many big data applications as it is able to compress it to a smaller file size making it easier to pull and analyze data from.

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.)

First, let us create a parquet file from the labevents.csv file

```r
labevents.parquet <- arrow::open_dataset("labevents.csv", format = "csv")
```

We see that the values updated after doing that in the environment. Let us create a parquet now.

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::write_dataset(arrow::open_dataset("labevents.csv", format="csv"),
                     path = "~/mimic/hosp/labevents", format = "parquet")
```

This is now creating a parquet of the previously opened dataset labevents.csv

```
ls -l ~/mimic/hosp/labevents
```

```
total 5341192
-rw-r--r--@ 1 lukehodges  staff  2731040379 Feb  7 15:49 part-0.parquet
```

Let us figure out the system time for the ingesting+selecting+filtering process. I am assuming that we do not include the arrow::write_dataset part as you said last time to do eval: false.

```
system.time({
  labevents.filter.parquet <- arrow::open_dataset("~/mimic/hosp/labevents",
                                                  format = "parquet") %>%
    dplyr::select(subject_id, itemid, charttime, valuenum) %>%
    dplyr::filter(itemid %in% c(50912, 50971, 50983, 50902, 50882, 51221,
                                51301, 50931))
})
```

```
   user  system elapsed
  0.468   0.114   0.585
```

The user, system, and elapsed times are 0.489, 0.094, and 0.605 respectively

Now we have to write the new dataset for the filtered.parquet. The directory will be in the labevents.filtered folder within the mimic

```
arrow::write_dataset(labevents.filter.parquet,
                     path = "~/mimic/hosp/labevents.filtered",
                     format = "parquet")
```

Check for file size now using the folder in which the filtered parquet is in

```
ls -l ~/mimic/hosp/labevents.filtered
```

```
total 327688
-rw-r--r--@ 1 lukehodges  staff  152917918 Feb  7 15:53 part-0.parquet
```

The file size is 152.9 MB after being filtered

Display the number of rows and the first 10 rows of the result tibble and make sure they match those in Q2.3.

```
nrow(labevents.filter.parquet)
```

```
[1] 32679896
```

This is the same amount as we saw before, 32679896

The first ten rows of the result tibble:

```
first10.parquet <- labevents.filter.parquet %>%
  head(10) %>%
  collect()
```

```
print(first10.parquet)
```

```
# 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  
```

This is the exact same as what we saw before

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

Parquet format creates efficient storage of data. It is used by storing and analyzing columns, rather than rows, which is far more effective and efficient. It is used in many big data applications as it is able to compress it to a smaller file size making it easier to pull and analyze data from. It reads individual columns.

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.

The first step is to ingest the parquet file

```
labevents.duckdb <-
  arrow::open_dataset("~/mimic/hosp/labevents/part-0.parquet",
                                        format = "parquet")
```

Then, we can make the new one again IF the table does not already exist. I received an error about dbplyr, so I installed it.

```
library(dbplyr)
```

Now, let us select columns and filter rows as in Q2.5 and see how long does the ingest+convert+select+filter process take?

```
system.time({
  labevents.filter.duckdb <-
    arrow::open_dataset("~/mimic/hosp/labevents/part-0.parquet",
                                               format = "parquet") %>%
    arrow::to_duckdb(table = "labevents.db") %>%
    dplyr::select(subject_id, itemid, charttime, valuenum) %>%
    dplyr::filter(itemid %in% c(50912, 50971, 50983, 50902, 50882, 51221,
                                 51301, 50931))
})
```

```
   user  system elapsed 
  0.549   0.120   0.679 
```

The user, system, and elapsed times are 0.509, 0.092, and 0.626, respectively. This is slower than both the arrow and paraquet methods

Display the number of rows and the first 10 rows of the result tibble and make sure they match those in Q2.3.

This command was taken and modified directly from the website attached to the homework for this section. We have to load dplyr since dbplyr overrode it

```r
library(dplyr)
```

```r
nrow(collect(labevents.filter.duckdb))
```

```
[1] 32679896
```

This is identical to what we have seen above: 32679896

First 10 rows of the result tibble

```r
first10.duckdb <- labevents.filter.duckdb %>%
  head(10) %>%
  collect()
```

```r
print(first10.duckdb)
```

```
# A tibble: 10 x 4
   subject_id itemid charttime           valuenum
        <dbl>  <dbl> <dttm>                 <dbl>
 1   10000980  51301 2191-05-23 10:20:00      4.6
 2   10000980  50882 2191-05-24 05:45:00     25  
 3   10000980  50902 2191-05-24 05:45:00    108  
 4   10000980  50912 2191-05-24 05:45:00      2.1
 5   10000980  50931 2191-05-24 05:45:00    116  
 6   10000980  50971 2191-05-24 05:45:00      4  
 7   10000980  50983 2191-05-24 05:45:00    144  
 8   10000980  51221 2191-05-24 05:45:00     28  
 9   10000980  51301 2191-05-24 05:45:00      3.4
10   10000980  51221 2191-05-30 12:40:00     28.8
```

This is identical to what we have seen above, with the time zones going back to what they were originally.

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

DuckDB, like Parquet and Arrow, utilizes a columnar based format which allows for easier access and storage of the data rather than using rows. It can read both Parquet and Arrow without loading the entire file into memory, which is beneficial for big data applications as it allows for faster turn around. It also does paralleling processing as mentioned by Dr. Hua Zhou in class.

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
```

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,,
```

How many rows? 433 millions.

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

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 > ~/mimic/icu/chartevents.csv
```

Make sure that the file is there:

```
ls -l ~/mimic/icu/chartevents.csv
```

```
-rw-r--r--@ 1 lukehodges  staff  41935806083 Feb  7 15:57 /Users/lukehodges/mimic/icu/charte
```

```
chartevents.arrow <- arrow::open_dataset("chartevents.csv", format = "csv")
```

Now we have to select the columns and filter the new table chartevents.filtered.arrow

```
chartevents.filtered.arrow <- chartevents.arrow %>%
  dplyr::select(subject_id, itemid, charttime, valuenum) %>%
  dplyr::filter(itemid %in% c(220045, 220181, 220179, 223761, 220210))
```

```
nrow(collect(chartevents.filtered.arrow))
```

```
[1] 30195426
```

There are 30195426 in chartevents.filtered.arrow

First ten lines of chartevents.filtered.arrow

```r
charteventsprint <- chartevents.filtered.arrow %>%
  head(10) %>%
  collect()
```

```r
print(charteventsprint)
```

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