

Biostat 203B Homework 1

Due Jan 24, 2024 @ 11:59PM

AUTHOR

Sophia Luo, 106409469

Display machine information for reproducibility:

```
sessionInfo()
```

R version 4.4.2 (2024-10-31)

Platform: x86_64-pc-linux-gnu

Running under: Ubuntu 24.04.1 LTS

Matrix products: default

BLAS: /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.12.0

LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.12.0

locale:

```
[1] LC_CTYPE=C.UTF-8      LC_NUMERIC=C           LC_TIME=C.UTF-8
[4] LC_COLLATE=C.UTF-8    LC_MONETARY=C.UTF-8    LC_MESSAGES=C.UTF-8
[7] LC_PAPER=C.UTF-8      LC_NAME=C              LC_ADDRESS=C
[10] LC_TELEPHONE=C        LC_MEASUREMENT=C.UTF-8 LC_IDENTIFICATION=C
```

time zone: America/Los_Angeles

tzcode source: system (glibc)

attached base packages:

```
[1] stats      graphics  grDevices  utils      datasets  methods    base
```

loaded via a namespace (and not attached):

```
[1] compiler_4.4.2  fastmap_1.2.0  cli_3.6.3      tools_4.4.2
[5] htmltools_0.5.8.1 yaml_2.3.10    rmarkdown_2.29 knitr_1.49
[9] jsonlite_1.8.9  xfun_0.50      digest_0.6.37  rlang_1.1.4
[13] evaluate_1.0.1
```

Q1. Git/GitHub

No handwritten homework reports are accepted for this course. We work with Git and GitHub. Efficient and abundant use of Git, e.g., frequent and well-documented commits, is an important criterion for grading your homework.

1. Apply for the [Student Developer Pack](#) at GitHub using your UCLA email. You'll get GitHub Pro account for free (unlimited public and private repositories).
2. Create a **private** repository [biostat-203b-2025-winter](#) and add [Hua-Zhou](#) and TA team ([Tomoki-Okuno](#) for Lec 1; [parsajamshidian](#) and [BowenZhang2001](#) for Lec 82) as your collaborators with write permission.

3. Top directories of the repository should be `hw1`, `hw2`, ... Maintain two branches `main` and `develop`. The `develop` branch will be your main playground, the place where you develop solution (code) to homework problems and write up report. The `main` branch will be your presentation area. Submit your homework files (Quarto file `qmd`, `html` file converted by Quarto, all code and extra data sets to reproduce results) in the `main` branch.
4. After each homework due date, course reader and instructor will check out your `main` branch for grading. Tag each of your homework submissions with tag names `hw1`, `hw2`, ... Tagging time will be used as your submission time. That means if you tag your `hw1` submission after deadline, penalty points will be deducted for late submission.
5. After this course, you can make this repository public and use it to demonstrate your skill sets on job market.

Solution: Done.

Q2. Data ethics training

This exercise (and later in this course) uses the [MIMIC-IV data v3.1](#), a freely accessible critical care database developed by the MIT Lab for Computational Physiology. Follow the instructions at <https://mimic.mit.edu/docs/gettingstarted/> to (1) complete the CITI [Data or Specimens Only Research](#) course and (2) obtain the PhysioNet credential for using the MIMIC-IV data. Display the verification links to your completion report and completion certificate here. **You must complete Q2 before working on the remaining questions.** (Hint: The CITI training takes a few hours and the PhysioNet credentialing takes a couple days; do not leave it to the last minute.)

Solution:

Data or Specimens Only Research

Completion report link: <https://www.citiprogram.org/verify/?kaffd7ce3-9d0f-44c1-9b2e-a8de96beb6e0-67196202>

Certification link: <https://www.citiprogram.org/verify/?w29f1696a-086a-41b4-8d37-4817c26dada5-67196202>

Conflicts of Interest Certificate

Completion report link: <https://www.citiprogram.org/verify/?k272a87e7-bbe9-40db-890a-cb80fbbb40ea-67196201>

Certification link: <https://www.citiprogram.org/verify/?w84033b44-4d38-4426-a7fb-552b1d7d6b87-67196201>

Q3. Linux Shell Commands

1. Make the MIMIC-IV v3.1 data available at location `~/mimic`. The output of the `ls -l ~/mimic` command should be similar to the below (from my laptop).

```
# content of mimic folder
ls -l ~/mimic/
```

```
total 24
```

```
-rwxrwxrwx 1 mmmm2627 mmmm2627 15199 Jan 16 12:39 CHANGELOG.txt
-rwxrwxrwx 1 mmmm2627 mmmm2627 2518 Jan 16 12:39 LICENSE.txt
-rwxrwxrwx 1 mmmm2627 mmmm2627 2884 Jan 16 12:39 SHA256SUMS.txt
drwxrwxrwx 1 mmmm2627 mmmm2627 4096 Jan 16 22:33 hosp
drwxrwxrwx 1 mmmm2627 mmmm2627 4096 Jan 16 13:45 icu
```

Refer to the documentation <https://physionet.org/content/mimiciv/3.1/> for details of data files. Do **not** put these data files into Git; they are big. Do **not** copy them into your directory. Do **not** decompress the gz data files. These create unnecessary big files and are not big-data-friendly practices. Read from the data folder `~/mimic` directly in following exercises.

Solution: I downloaded the MIMIC-IV data and it's available under `~/mimic`.

Use Bash commands to answer following questions.

2. Display the contents in the folders `hosp` and `icu` using Bash command `ls -l`. Why are these data files distributed as `.csv.gz` files instead of `.csv` (comma separated values) files? Read the page <https://mimic.mit.edu/docs/iv/> to understand what's in each folder.

Solution: Content of `hosp` folder:

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

```
total 24124660
```

```
-rwxrwxrwx 1 mmmm2627 mmmm2627 19928140 Jan 16 12:39 admissions.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 427554 Jan 16 12:39 d_hcpcs.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 876360 Jan 16 12:39 d_icd_diagnoses.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 589186 Jan 16 12:39 d_icd_procedures.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 13169 Jan 16 12:39 d_labitems.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 33564802 Jan 16 12:39 diagnoses_icd.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 9743908 Jan 16 12:39 drgcodes.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 811305629 Jan 16 12:39 emar.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 748158322 Jan 16 12:39 emar_detail.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 2162335 Jan 16 12:39 hcpcsevents.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 18402851720 Jan 16 12:39 labevents.csv
-rwxrwxrwx 1 mmmm2627 mmmm2627 2592909134 Jan 16 12:39 labevents.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 117644075 Jan 16 12:39 microbiologyevents.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 44069351 Jan 16 12:39 omr.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 2835586 Jan 16 12:39 patients.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 525708076 Jan 16 12:39 pharmacy.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 666594177 Jan 16 12:39 poe.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 55267894 Jan 16 12:39 poe_detail.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 606298611 Jan 16 12:39 prescriptions.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 7777324 Jan 16 12:39 procedures_icd.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 127330 Jan 16 12:39 provider.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 8569241 Jan 16 12:39 services.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 46185771 Jan 16 12:39 transfers.csv.gz
```

Content of `icu` folder:

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

```
total 4253392
-rwxrwxrwx 1 mmmm2627 mmmm2627      41566 Jan 16 12:39 caregiver.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 3502392765 Jan 16 12:40 chartevents.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627      58741 Jan 16 12:40 d_items.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 63481196 Jan 16 12:40 datatimeevents.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627   3342355 Jan 16 12:40 icustays.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 311642048 Jan 16 12:40 ingredientevents.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 401088206 Jan 16 12:40 inputevents.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 49307639 Jan 16 12:40 outputevents.csv.gz
-rwxrwxrwx 1 mmmm2627 mmmm2627 24096834 Jan 16 12:40 procedureevents.csv.gz
```

The extension **gz** means that the file is compressed. The data files have large sizes, so they have to be zipped to be downloaded faster.

3. Briefly describe what Bash commands **zcat**, **zless**, **zmore**, and **zgrep** do.

Solution:

All four commands applies to files with **.gz** extension. **zcat** displays the contents of compressed files without decompressing them to a separate file. **zless** views the contents of compressed files in a scrollable viewer, similar to **less**. **zmore** views the contents of compressed files in a paginated way (one screenful at a time), similar to **more**. **zgrep** searches for a specified pattern within compressed files and outputs matching lines, similar to **grep**.

4. (Looping in Bash) What's the output of the following bash script?

```
for datafile in ~/mimic/hosp/{a,l,pa}*.gz
do
  ls -l $datafile
done
```

Solution: The output shows all files with names beginning with **a**, **l**, or **pa** under directory **hosp**.

Display the number of lines in each data file using a similar loop. (Hint: combine linux commands **zcat** < and **wc -l**.)

Solution:

Note: I use **cache** to display output from previous running to reduce the rendering time.

```
for datafile in ~/mimic/*/*.gz
do
  echo "$datafile: $(zcat < $datafile | wc -l)"
done
```

```
/home/mmmm2627/mimic/hosp/admissions.csv.gz: 546029
/home/mmmm2627/mimic/hosp/d_hcpcs.csv.gz: 89209
/home/mmmm2627/mimic/hosp/d_icd_diagnoses.csv.gz: 112108
```

```

/home/mmmm2627/mimic/hosp/d_icd_procedures.csv.gz: 86424
/home/mmmm2627/mimic/hosp/d_labitems.csv.gz: 1651
/home/mmmm2627/mimic/hosp/diagnoses_icd.csv.gz: 6364489
/home/mmmm2627/mimic/hosp/drgcodes.csv.gz: 761857
/home/mmmm2627/mimic/hosp/emar.csv.gz: 42808594
/home/mmmm2627/mimic/hosp/emar_detail.csv.gz: 87371065
/home/mmmm2627/mimic/hosp/hcpcsevents.csv.gz: 186075
/home/mmmm2627/mimic/hosp/labevents.csv.gz: 158374765
/home/mmmm2627/mimic/hosp/microbiologyevents.csv.gz: 3988225
/home/mmmm2627/mimic/hosp/omr.csv.gz: 7753028
/home/mmmm2627/mimic/hosp/patients.csv.gz: 364628
/home/mmmm2627/mimic/hosp/pharmacy.csv.gz: 17847568
/home/mmmm2627/mimic/hosp/poe.csv.gz: 52212110
/home/mmmm2627/mimic/hosp/poe_detail.csv.gz: 8504983
/home/mmmm2627/mimic/hosp/prescriptions.csv.gz: 20292612
/home/mmmm2627/mimic/hosp/procedures_icd.csv.gz: 859656
/home/mmmm2627/mimic/hosp/provider.csv.gz: 42245
/home/mmmm2627/mimic/hosp/services.csv.gz: 593072
/home/mmmm2627/mimic/hosp/transfers.csv.gz: 2413582
/home/mmmm2627/mimic/icu/caregiver.csv.gz: 17985
/home/mmmm2627/mimic/icu/chartevents.csv.gz: 432997492
/home/mmmm2627/mimic/icu/d_items.csv.gz: 4096
/home/mmmm2627/mimic/icu/datetimeevents.csv.gz: 9979762
/home/mmmm2627/mimic/icu/icustays.csv.gz: 94459
/home/mmmm2627/mimic/icu/ingredientevents.csv.gz: 14253481
/home/mmmm2627/mimic/icu/inputevents.csv.gz: 10953714
/home/mmmm2627/mimic/icu/outputevents.csv.gz: 5359396
/home/mmmm2627/mimic/icu/procedureevents.csv.gz: 808707

```

5. Display the first few lines of `admissions.csv.gz`. How many rows are in this data file, excluding the header line? Each `hadm_id` identifies a hospitalization. How many hospitalizations are in this data file? How many unique patients (identified by `subject_id`) are in this data file? Do they match the number of patients listed in the `patients.csv.gz` file? (Hint: combine Linux commands `zcat <`, `head/tail`, `awk`, `sort`, `uniq`, `wc`, and so on.)

Solution:

The first few lines of `admissions.csv.gz`:

```
zcat < ~/mimic/hosp/admissions.csv.gz | head
```

```

subject_id,hadm_id,admittime,dischtime,deathtime,admission_type,admit_provider_id,admission_location,discharge_location,insurance,language,marital_status,race,edregtime,edouttime,hospital_expire_flag
10000032,22595853,2180-05-06 22:23:00,2180-05-07 17:15:00,,URGENT,P49AFC,TRANSFER FROM HOSPITAL,HOME,Medicaid,English,WIDOWED,WHITE,2180-05-06 19:17:00,2180-05-06 23:30:00,0
10000032,22841357,2180-06-26 18:27:00,2180-06-27 18:49:00,,EW EMER.,P784FA,EMERGENCY ROOM,HOME,Medicaid,English,WIDOWED,WHITE,2180-06-26 15:54:00,2180-06-26 21:31:00,0
10000032,25742920,2180-08-05 23:44:00,2180-08-07 17:50:00,,EW EMER.,P19UTS,EMERGENCY ROOM,HOSPICE,Medicaid,English,WIDOWED,WHITE,2180-08-05 20:58:00,2180-08-06 01:44:00,0

```

```

10000032,29079034,2180-07-23 12:35:00,2180-07-25 17:55:00,,EW EMER.,P060TX,EMERGENCY
ROOM,HOME,Medicaid,English,WIDOWED,WHITE,2180-07-23 05:54:00,2180-07-23 14:00:00,0
10000068,25022803,2160-03-03 23:16:00,2160-03-04 06:26:00,,EU OBSERVATION,P39NWO,EMERGENCY
ROOM,,,English,SINGLE,WHITE,2160-03-03 21:55:00,2160-03-04 06:26:00,0
10000084,23052089,2160-11-21 01:56:00,2160-11-25 14:52:00,,EW EMER.,P42H7G,WALK-IN/SELF
REFERRAL,HOME HEALTH CARE,Medicare,English,MARRIED,WHITE,2160-11-20 20:36:00,2160-11-21
03:20:00,0
10000084,29888819,2160-12-28 05:11:00,2160-12-28 16:07:00,,EU OBSERVATION,P35NE4,PHYSICIAN
REFERRAL,,Medicare,English,MARRIED,WHITE,2160-12-27 18:32:00,2160-12-28 16:07:00,0
10000108,27250926,2163-09-27 23:17:00,2163-09-28 09:04:00,,EU OBSERVATION,P40JML,EMERGENCY
ROOM,,,English,SINGLE,WHITE,2163-09-27 16:18:00,2163-09-28 09:04:00,0
10000117,22927623,2181-11-15 02:05:00,2181-11-15 14:52:00,,EU OBSERVATION,P47EY8,EMERGENCY
ROOM,,Medicaid,English,DIVORCED,WHITE,2181-11-14 21:51:00,2181-11-15 09:57:00,0

```

The number of rows in this data file, excluding the header line:

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

546028

Note:

- `uniq` detects duplicates only if they are next to each other, so we need to `sort` first.
- `tail -n +2` is used to exclude the header.

The number of hospitalizations in this data file:

```

zcat < ~/mimic/hosp/admissions.csv.gz |
tail -n +2 |
cut -d, -f2 |
sort |
uniq |
wc -l

```

546028

Alternatively using `awk`:

```

zcat < ~/mimic/hosp/admissions.csv.gz |
tail -n +2 |
awk -F, '{print $2}' |
sort |
uniq |
wc -l

```

546028

Peek the first few lines of `patients.csv.gz`:

```
zcat < ~/mimic/hosp/patients.csv.gz | head
```

```
subject_id,gender,anchor_age,anchor_year,anchor_year_group,dod
10000032,F,52,2180,2014 - 2016,2180-09-09
10000048,F,23,2126,2008 - 2010,
10000058,F,33,2168,2020 - 2022,
10000068,F,19,2160,2008 - 2010,
10000084,M,72,2160,2017 - 2019,2161-02-13
10000102,F,27,2136,2008 - 2010,
10000108,M,25,2163,2014 - 2016,
10000115,M,24,2154,2017 - 2019,
10000117,F,48,2174,2008 - 2010,
```

The number of unique patients in `admissions.csv.gz` is

```
zcat < ~/mimic/hosp/admissions.csv.gz |
tail -n +2 |
awk -F, '{print $1}' |
sort |
uniq |
wc -l
```

223452

which is less than the number of patients listed in the `patients.csv.gz` file:

```
zcat < ~/mimic/hosp/patients.csv.gz |
tail -n +2 |
awk -F, '{print $1}' |
sort |
uniq |
wc -l
```

364627

6. What are the possible values taken by each of the variable `admission_type`, `admission_location`, `insurance`, and `ethnicity`? Also report the count for each unique value of these variables in decreasing order. (Hint: combine Linux commands `zcat`, `head` / `tail`, `awk`, `uniq -c`, `wc`, `sort`, and so on; skip the header line.)

Solution:

The possible values taken by `admission_type` and the count in decreasing order:

```
zcat < ~/mimic/hosp/admissions.csv.gz |
tail -n +2 |
awk -F, '{print $6}' |
sort |
```

```
uniq -c |
sort -r
```

```
177459 EW EMER.
119456 EU OBSERVATION
84437 OBSERVATION ADMIT
54929 URGENT
42898 SURGICAL SAME DAY ADMISSION
24551 DIRECT OBSERVATION
21973 DIRECT EMER.
13130 ELECTIVE
7195 AMBULATORY OBSERVATION
```

The possible values taken by `admission_location` and the count in decreasing order:

```
zcat < ~/mimic/hosp/admissions.csv.gz |
tail -n +2 |
awk -F, '{print $8}' |
sort |
uniq -c |
sort -r
```

```
244179 EMERGENCY ROOM
163228 PHYSICIAN REFERRAL
56227 TRANSFER FROM HOSPITAL
42365 WALK-IN/SELF REFERRAL
12965 CLINIC REFERRAL
8518 PROCEDURE SITE
6317 TRANSFER FROM SKILLED NURSING FACILITY
5837 INTERNAL TRANSFER TO OR FROM PSYCH
5734 PACU
402 INFORMATION NOT AVAILABLE
255 AMBULATORY SURGERY TRANSFER
1
```

The possible values taken by `insurance` and the count in decreasing order:

```
zcat < ~/mimic/hosp/admissions.csv.gz |
tail -n +2 |
awk -F, '{print $10}' |
sort |
uniq -c |
sort -r
```

```
244576 Medicare
173399 Private
104229 Medicaid
14006 Other
9355
463 No charge
```


The possible values taken by **ethnicity** (**race** in the file) and the count in decreasing order:

```
zcat < ~/mimic/hosp/admissions.csv.gz |
tail -n +2 |
awk -F, '{print $13}' |
sort |
uniq -c |
sort -r
```

```
336538 WHITE
75482 BLACK/AFRICAN AMERICAN
19788 OTHER
13972 WHITE - OTHER EUROPEAN
13870 UNKNOWN
10903 HISPANIC/LATINO - PUERTO RICAN
8287 HISPANIC OR LATINO
7809 ASIAN
7644 ASIAN - CHINESE
6597 WHITE - RUSSIAN
6205 BLACK/CAPE VERDEAN
6070 HISPANIC/LATINO - DOMINICAN
3875 BLACK/CARIBBEAN ISLAND
3495 BLACK/AFRICAN
3478 UNABLE TO OBTAIN
2162 PATIENT DECLINED TO ANSWER
2082 PORTUGUESE
1973 ASIAN - SOUTH EAST ASIAN
1886 WHITE - EASTERN EUROPEAN
1858 HISPANIC/LATINO - GUATEMALAN
1661 ASIAN - ASIAN INDIAN
1526 WHITE - BRAZILIAN
1320 HISPANIC/LATINO - SALVADORAN
1247 AMERICAN INDIAN/ALASKA NATIVE
920 HISPANIC/LATINO - COLUMBIAN
883 HISPANIC/LATINO - MEXICAN
774 SOUTH AMERICAN
725 HISPANIC/LATINO - HONDURAN
664 ASIAN - KOREAN
641 HISPANIC/LATINO - CUBAN
603 HISPANIC/LATINO - CENTRAL AMERICAN
596 MULTIPLE RACE/ETHNICITY
494 NATIVE HAWAIIAN OR OTHER PACIFIC ISLANDER
```

7. The **icusays.csv.gz** file contains all the ICU stays during the study period. How many ICU stays, identified by **stay_id**, are in this data file? How many unique patients, identified by **subject_id**, are in this data file?

Solution:

Peek the first few lines of the file:

```
zcat < ~/mimic/icu/icustays.csv.gz | head
```

```
subject_id,hadm_id,stay_id,first_careunit,last_careunit,intime,outtime,los
10000032,29079034,39553978,Medical Intensive Care Unit (MICU),Medical Intensive Care Unit
(MICU),2180-07-23 14:00:00,2180-07-23 23:50:47,0.4102662037037037
10000690,25860671,37081114,Medical Intensive Care Unit (MICU),Medical Intensive Care Unit
(MICU),2150-11-02 19:37:00,2150-11-06 17:03:17,3.8932523148148146
10000980,26913865,39765666,Medical Intensive Care Unit (MICU),Medical Intensive Care Unit
(MICU),2189-06-27 08:42:00,2189-06-27 20:38:27,0.4975347222222222
10001217,24597018,37067082,Surgical Intensive Care Unit (SICU),Surgical Intensive Care Unit
(SICU),2157-11-20 19:18:02,2157-11-21 22:08:00,1.1180324074074075
10001217,27703517,34592300,Surgical Intensive Care Unit (SICU),Surgical Intensive Care Unit
(SICU),2157-12-19 15:42:24,2157-12-20 14:27:41,0.948113425925926
10001725,25563031,31205490,Medical/Surgical Intensive Care Unit (MICU/SICU),Medical/Surgical
Intensive Care Unit (MICU/SICU),2110-04-11 15:52:22,2110-04-12 23:59:56,1.338587962962963
10001843,26133978,39698942,Medical/Surgical Intensive Care Unit (MICU/SICU),Medical/Surgical
Intensive Care Unit (MICU/SICU),2134-12-05 18:50:03,2134-12-06 14:38:26,0.8252662037037037
10001884,26184834,37510196,Medical Intensive Care Unit (MICU),Medical Intensive Care Unit
(MICU),2131-01-11 04:20:05,2131-01-20 08:27:30,9.17181712962963
10002013,23581541,39060235,Cardiac Vascular Intensive Care Unit (CVICU),Cardiac Vascular
Intensive Care Unit (CVICU),2160-05-18 10:00:53,2160-05-19 17:33:33,1.314351851851852
```

The number of ICU stays identified by `stay_id`:

```
zcat < ~/mimic/icu/icustays.csv.gz |
tail -n +2 |
awk -F, '{print $3}' |
sort |
uniq |
wc -l
```

94458

The number of unique patients identified by `subject_id`:

```
zcat < ~/mimic/icu/icustays.csv.gz |
tail -n +2 |
awk -F, '{print $1}' |
sort |
uniq |
wc -l
```

65366

8. *To compress, or not to compress. That's the question.* Let's focus on the big data file `labevents.csv.gz`. Compare compressed gz file size to the uncompressed file size. Compare the run times of `zcat < ~/mimic/labevents.csv.gz | wc -l` versus `wc -l labevents.csv`. Discuss the trade off between storage

and speed for big data files. (Hint: `gzip -dk < FILENAME.gz > ./FILENAME`. Remember to delete the large `labevents.csv` file after the exercise.)

Solution:

I runned `gzip -dk labevents.csv.gz ./labevents.csv` to decompress the file into `labevents.csv`.

File size comparison:

```
ls -lh ~/mimic/hosp/labevents*
```

```
-rwxrwxrwx 1 mmmm2627 mmmm2627 18G Jan 16 12:39 /home/mmmm2627/mimic/hosp/labevents.csv
-rwxrwxrwx 1 mmmm2627 mmmm2627 2.5G Jan 16 12:39 /home/mmmm2627/mimic/hosp/labevents.csv.gz
```

The uncompressed file (18G) is more than 7 times larger than the compressed file (2.5G).

Note: I use `cache` to display output from previous running to reduce the rendering time.

The run time of `zcat | wc` on compressed file:

```
time zcat < ~/mimic/hosp/labevents.csv.gz | wc -l
```

```
158374765
```

```
real    1m23.392s
user    0m56.314s
sys     0m9.535s
```

The run time of `wc` on uncompressed file:

```
time wc -l ~/mimic/hosp/labevents.csv
```

```
158374765 /home/mmmm2627/mimic/hosp/labevents.csv
```

```
real    4m0.895s
user    0m0.490s
sys     0m3.850s
```

The runtime of compressed file is 1 min 38 secs, compared to uncompressed file of 4m 28 secs.

The expected tradeoff is to balance storage space and run time. Compressed files are much smaller than the uncompressed counterparts, significantly reducing storage requirement. However, operating on compressed files is slower because the file must be decompressed first. The uncompressed files operation is faster since the file is directly accessible without decompression, but it takes lots of storage space.

My result deviates from the expectation, and here's my thoughts of the potential reasons:

- Disk I/O Bottleneck: The file needs to be read from disk. For large uncompressed files, the time it takes to read the file from disk dominates the runtime. Even though there's no decompression step, the sheer volume of

data in the uncompressed file makes the operation slower.

- b. Efficient Decompression: Gzip algorithms are highly optimized for decompression, so the CPU overhead for decompression might be negligible compared to the disk I/O savings.

The large uncompressed file is deleted using the command `rm ~/mimic/hosp/labevents.csv`.

Q4. Who's popular in Price and Prejudice

1. You and your friend just have finished reading *Pride and Prejudice* by Jane Austen. Among the four main characters in the book, Elizabeth, Jane, Lydia, and Darcy, your friend thinks that Darcy was the most mentioned. You, however, are certain it was Elizabeth. Obtain the full text of the novel from <http://www.gutenberg.org/cache/epub/42671/pg42671.txt> and save to your local folder.

```
wget -nc http://www.gutenberg.org/cache/epub/42671/pg42671.txt
```

Explain what `wget -nc` does. Do **not** put this text file `pg42671.txt` in Git. Complete the following loop to tabulate the number of times each of the four characters is mentioned using Linux commands.

Solution:

`wget` downloads the files from the web using the link provided by users. `-nc` specifies that `wget` will not overwrite an existing file of the same name in the current directory. Therefore, when the file is already downloaded, the content won't be retrieved again.

```
wget -nc http://www.gutenberg.org/cache/epub/42671/pg42671.txt
for char in Elizabeth Jane Lydia Darcy
do
    count=$(grep -o "$char" "pg42671.txt" | wc -l)
    echo "$char: $count"
done
```

2. What's the difference between the following two commands?

```
echo 'hello, world' > test1.txt
```

and

```
echo 'hello, world' >> test2.txt
```

Solution:

Both `>` and `>>` are redirection operators. They both create the file if it doesn't exist. The main difference is that when running the command again while content exist, `>` will overwrite existing content but `>>` will preserve existing content and adds new content at the end. For example, after running the above command for 4 times, `test1.txt` will only have "hello, world" in first line, but `test2.txt` will have 4 "hello, world" in 4 lines.

3. Using your favorite text editor (e.g., `vi`), type the following and save the file as `middle.sh`:

```
#!/bin/sh
# Select lines from the middle of a file.
# Usage: bash middle.sh filename end_line num_lines
head -n "$2" "$1" | tail -n "$3"
```

Using `chmod` to make the file executable by the owner, and run

```
./middle.sh pg42671.txt 20 5
```

Explain the output. Explain the meaning of `"$1"`, `"$2"`, and `"$3"` in this shell script. Why do we need the first line of the shell script?

Solution:

When executing `./middle.sh pg42671.txt 20 5`, the `.sh` file is executed with three parameters passed in:

`$1: pg42671.txt`

`$2: 20`

`$3: 5`

Then in `middle.sh` file, when `head` and `tail` command are run, they look for the actual `$1`, `$2`, `$3` sent along with the file. This is similar to running a function, where we specifies the parameters when calling the function, and within the function, we use temporary variable names instead of the actual variables.

The first line of the shell script `#!/bin/sh` is shebang or hashbang, and it specifies which shell or interpreter should execute the script. Here, it tells the system to use the Bourne shell `sh` to execute the script.

Q5. More fun with Linux

Try following commands in Bash and interpret the results: `cal`, `cal 2025`, `cal 9 1752` (anything unusual?), `date`, `hostname`, `arch`, `uname -a`, `uptime`, `who am i`, `who`, `w`, `id`, `last | head`, `echo {con,pre}{sent,fer}{s,ed}`, `time sleep 5`, `history | tail`.

Solution:

```
cal
```

```

January 2025
Su Mo Tu We Th Fr Sa
    1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
```

26 27 28 29 30 31

`cal` : prints out calender of current month.`cal 2025`

```

                2025
    January      February      March
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
      1  2  3  4              1
5  6  7  8  9 10 11  2  3  4  5  6  7  8  2  3  4  5  6  7  8
12 13 14 15 16 17 18  9 10 11 12 13 14 15  9 10 11 12 13 14 15
19 20 21 22 23 24 25 16 17 18 19 20 21 22 16 17 18 19 20 21 22
26 27 28 29 30 31    23 24 25 26 27 28    23 24 25 26 27 28 29
                                30 31

    April        May          June
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
      1  2  3  4  5              1  2  3  1  2  3  4  5  6  7
6  7  8  9 10 11 12  4  5  6  7  8  9 10  8  9 10 11 12 13 14
13 14 15 16 17 18 19 11 12 13 14 15 16 17 15 16 17 18 19 20 21
20 21 22 23 24 25 26 18 19 20 21 22 23 24 22 23 24 25 26 27 28
27 28 29 30    25 26 27 28 29 30 31 29 30

    July         August       September
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
      1  2  3  4  5              1  2  1  2  3  4  5  6
6  7  8  9 10 11 12  3  4  5  6  7  8  9  7  8  9 10 11 12 13
13 14 15 16 17 18 19 10 11 12 13 14 15 16 14 15 16 17 18 19 20
20 21 22 23 24 25 26 17 18 19 20 21 22 23 21 22 23 24 25 26 27
27 28 29 30 31    24 25 26 27 28 29 30 28 29 30
                31

    October      November     December
Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa Su Mo Tu We Th Fr Sa
      1  2  3  4              1  1  2  3  4  5  6
5  6  7  8  9 10 11  2  3  4  5  6  7  8  7  8  9 10 11 12 13
12 13 14 15 16 17 18  9 10 11 12 13 14 15 14 15 16 17 18 19 20
19 20 21 22 23 24 25 16 17 18 19 20 21 22 21 22 23 24 25 26 27
26 27 28 29 30 31    23 24 25 26 27 28 29 28 29 30 31
                30

```

`cal 2025` : prints out calender of 2025.`cal 9 1752`

```

September 1752
Su Mo Tu We Th Fr Sa

```

```
    1  2 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

`cal 9 1752` : prints out calendar of September of 1752 but it's missing 3rd to 13th.

```
date
```

```
Sat Jan 18 23:35:59 PST 2025
```

`date` : prints out current date, time, and time zone.

```
hostname
```

```
Sophia-Laptop
```

`hostname` : prints out name of the host.

```
arch
```

```
x86_64
```

`arch` shows machine architecture of the system, including hardware platform and processor architecture. Here it shows that my laptop is using 64-bit architecture.

```
uname -a
```

```
Linux Sophia-Laptop 5.15.167.4-microsoft-standard-WSL2 #1 SMP Tue Nov 5 00:21:55 UTC 2024 x86_64
x86_64 x86_64 GNU/Linux
```

`uname` displays all available information about my system.

```
uptime
```

```
23:35:59 up 30 min,  1 user,  load average: 0.23, 0.09, 0.20
```

`uptime` shows how long the system has been running, along with information about the system's load averages. Here it shows that the system has been up for over 3 hours and 30 minutes.

```
whoami
```

```
mmm2627
```

`whoami` shows the current logged-in user's username.

```
who
```

```
mmmm2627 pts/1          2025-01-18 23:03
```

`who` displays information about the currently logged-in user, the terminal, and the time at which the user logged in.

```
w
```

```
23:35:59 up 30 min,  1 user,  load average: 0.23, 0.09, 0.20
USER      TTY      FROM            LOGIN@   IDLE   JCPU   PCPU   WHAT
mmmm2627 pts/1    -                23:03   32:04   0.01s  0.01s  -bash
```

`w` provides detailed information about the users currently logged in, activities, and system information such as uptime and load averages.

```
id
```

```
uid=1000(mmmm2627) gid=1000(mmmm2627)
groups=1000(mmmm2627),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo),29(audio),30(dip),44(video),46(plugdev),100(users),107(netdev)
```

`id` displays the user and group information for the current user or a specified user.

```
last | head
```

```
reboot    system boot  5.15.167.4-micro Sat Jan 18 23:03    still running
reboot    system boot  5.15.167.4-micro Sat Jan 18 20:35    still running
reboot    system boot  5.15.167.4-micro Fri Jan 17 15:01    still running
reboot    system boot  5.15.167.4-micro Thu Jan 16 14:00    still running
reboot    system boot  5.15.167.4-micro Thu Jan 16 13:54    still running
reboot    system boot  5.15.167.4-micro Thu Jan 16 12:37    still running
reboot    system boot  5.15.167.4-micro Wed Jan 15 11:00    still running
reboot    system boot  5.15.167.4-micro Tue Jan 14 15:06    still running
reboot    system boot  5.15.167.4-micro Tue Jan  7 21:42    still running
reboot    system boot  5.15.167.4-micro Tue Jan  7 21:41    still running
```

`last` shows a list of the most recent logins on the system.

```
echo {con,pre}{sent,fer}{s,ed}
```

consents consented confers conferred presents presented prefers preferred

This prints out all combinations of contents in the three brackets in order.

```
time sleep 5
```



```
real    0m5.001s
user    0m0.001s
sys     0m0.000s
```

`sleep` pauses the execution of a script or command for a specified duration, in this case, 5 secs. `time` shows that `sleep 5` took about 6 secs to run, so the execution did pause for 5 secs.

`history` display the most recent commands that have been executed in the current terminal session..

Q6. Book

1. Git clone the repository <https://github.com/christophergandrud/Rep-Res-Book> for the book *Reproducible Research with R and RStudio* to your local machine. Do **not** put this repository within your homework repository `biostat-203b-2025-winter`.
2. Open the project by clicking `rep-res-3rd-edition.Rproj` and compile the book by clicking `Build Book` in the `Build` panel of RStudio. (Hint: I was able to build `git_book` and `epub_book` directly. For `pdf_book`, I needed to add a line `\usepackage{hyperref}` to the file `Rep-Res-Book/rep-res-3rd-edition/latex/preabmle.tex`.)

The point of this exercise is (1) to obtain the book for free and (2) to see an example how a complicated project such as a book can be organized in a reproducible way. Use `sudo apt install PKGNAME` to install required Ubuntu packages and `tlmgr install PKGNAME` to install missing TeXLive packages.

For grading purpose, include a screenshot of Section 4.1.5 of the book here.

Solution:

I have cloned the repository in a separate folder, installed the relevant packages, and compiled the book. Here's the screenshot of section 4.1.5:

Appendix: knitr and Lyx

4 Getting Started with File Managem...

4.1 File Paths and Naming Conve...

4.1.1 Root directories

4.1.2 Sub-directories and pare...

4.1.3 Working directories

4.1.4 Absolute vs. relative paths

4.1.5 Spaces in directory and f...

4.2 Organizing Your Research Pr...

4.3 Organizing Research with RSt...

4.4 R File Manipulation Functions

4.5 Unix-like Shell Commands for...

4.6 File Navigation in RStudio

Chapter summary

II Data Gathering and Storage

5 Storing, Collaborating, Accessing F...

5.1 Saving Data in Reproducible ...

5.2 Storing Your Files in the Clou...

5.2.1 Storage

5.2.2 Accessing data

5.2.3 Collaboration

4.1.5 Spaces in directory and file names

It is good practice to avoid putting spaces in your file and directory names. For example, I called the example project parent directory in Figure 4.1 "example-project" rather than "Example Project". Spaces in file and directory names can sometimes create problems for computer programs trying to read the file path. The program may believe that the space indicates that the path name has ended. To make multi-word names easily readable without using spaces, adopt a consistent naming convention.

One approach is to use a convention that contrasts with the R object naming convention you are using. A contrasting convention helps make it clear if something is an R object or a file name. For example, if we adopt the underscore method for R object names used in Chapter 3 (e.g. `health_data`) we could use hyphens (-) to separate words in file names. For example: `example-source.R`. This is sometimes called kebab-case.

4.2 Organizing Your Research Project

Figure 4.1 gives an example of how the files in a simple reproducible research project could be organized. The project's parent directory is called *example-project*. Inside this directory are the primary knittable documents (*paper.Rmd*, *slideshow.Rmd*, and *website.Rmd*). In addition, there is an *analysis* sub-directory with the R files to run the statistical analyses followed by a further *data* child directory.

The nested file structure allows you to use relative file paths. The knittable documents can call *analysis-1.R* with the relative path *analysis/analysis-1.R*.

Project: (None)

Section 4.1.5