Biostat 203B Homework 1

Due Jan 24, 2025 @ 11:59PM

Zhiyuan Yu 906405523

Display machine information for reproducibility:

```
sessionInfo()
```

```
R version 4.4.2 (2024-10-31)
Platform: x86_64-pc-linux-gnu
Running under: Ubuntu 24.04.1 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.12.0
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.12.0

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

time zone: America/Los_Angeles
tzcode source: system (glibc)

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

loaded via a namespace (and not attached):
 [1] 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_1.0.3
```

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 Here is the Completion Report and Completion Certificate of my CITI training.

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 zxhyu zxhyu 15199 Jan 21 23:40 CHANGELOG.txt
-rwxrwxrwx 1 zxhyu zxhyu  2518 Jan 21 23:40 LICENSE.txt
-rwxrwxrwx 1 zxhyu zxhyu  2884 Jan 21 23:40 SHA256SUMS.txt
drwxrwxrwx 1 zxhyu zxhyu  4096 Jan 24 00:49 hosp
drwxrwxrwx 1 zxhyu zxhyu  4096 Jan 21 23:42 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.

Use Bash commands to answer following questions.

Solution: I have downloaded the MIMIC IV v3.1 data and it's available under `~/mimic` folder as requested.

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: Here is the content of `hosp` folder

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

```
total 6153124
-rwxrwxrwx 1 zxhyu zxhyu   19928140 Jan 21 23:40 admissions.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu     427554 Jan 21 23:40 d_hcpcs.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu     876360 Jan 21 23:40 d_icd_diagnoses.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu     589186 Jan 21 23:40 d_icd_procedures.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu      13169 Jan 21 23:40 d_labitems.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu   33564802 Jan 21 23:40 diagnoses_icd.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu    9743908 Jan 21 23:40 drgcodes.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu  811305629 Jan 21 23:40 emar.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu  748158322 Jan 21 23:40 emar_detail.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu    2162335 Jan 21 23:40 hcpcsevents.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu 2592909134 Jan 21 23:41 labevents.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu  117644075 Jan 21 23:41 microbiologyevents.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu   44069351 Jan 21 23:41 omr.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu    2835586 Jan 21 23:41 patients.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu  525708076 Jan 21 23:41 pharmacy.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu  666594177 Jan 21 23:41 poe.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu   55267894 Jan 21 23:41 poe_detail.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu  606298611 Jan 21 23:41 prescriptions.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu    7777324 Jan 21 23:41 procedures_icd.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu     127330 Jan 21 23:41 provider.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu    8569241 Jan 21 23:41 services.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu   46185771 Jan 21 23:41 transfers.csv.gz
```

and the content of the `icu` folder

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

```
total 4253392
-rwxrwxrwx 1 zxhyu zxhyu      41566 Jan 21 23:41 caregiver.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu 3502392765 Jan 21 23:42 chartevents.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu      58741 Jan 21 23:42 d_items.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu   63481196 Jan 21 23:42 datetimeevents.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu    3342355 Jan 21 23:42 icustays.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu  311642048 Jan 21 23:42 ingredientevents.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu  401088206 Jan 21 23:42 inputevents.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu   49307639 Jan 21 23:42 outputevents.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu   24096834 Jan 21 23:42 procedureevents.csv.gz
```

The .csv.gz files are compressed versions of .csv files. These data were distributed as `.csv.gz` file because the files are too large to be stored or downloaded. To ensure there are enough spaces to store and the users could downlad the dataset in faster speed, the data files are distributed as `.csv.gz` files instead of the `.csv`

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

Solution: For `zcat`, it reads and outputs the contents of a .gz file to the standard output; for `zless`, it opens a .gz file in a scrollable viewer and allows you to navigate the file's content without extracting the file; for `zmore`, it displays the content of a .gz file one screen at a time; for `zgrep`, it looks for specific patterns or keywords within a .gz file.

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
```

```
-rwxrwxrwx  1  zxhyu  zxhyu  19928140  Jan  21  23:40  /home/zxhyu/mimic/hosp/
admissions.csv.gz
-rwxrwxrwx  1  zxhyu  zxhyu  2592909134  Jan  21  23:41  /home/zxhyu/mimic/hosp/
labevents.csv.gz
-rwxrwxrwx  1  zxhyu  zxhyu  2835586  Jan  21  23:41  /home/zxhyu/mimic/hosp/
patients.csv.gz
```

Solution: The above is the output of the bash script.

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

```
for datafile in ~/mimic/hosp/{a,l,pa}*.gz
do
  echo "File: $datafile"
```

```
  zcat "$datafile" | wc -l
done
```

```
File: /home/zxhyu/mimic/hosp/admissions.csv.gz
546029
File: /home/zxhyu/mimic/hosp/labevents.csv.gz
158374765
File: /home/zxhyu/mimic/hosp/patients.csv.gz
364628
```

Solution: The number of lines in admissions is 546029, the number of lines in labevents is 158374765, the number of lines in patients is 364628.

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: Here is 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_locat
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.,P06OTX,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, is

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

```
546028
```

The number of hospitalizations in this data file is

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

```
546028
```

which is the same as the number of rows in the file.

The number of unique patients in this data file 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:

```
zcat ~/mimic/hosp/admissions.csv.gz | head -1
```

```
subject_id,hadm_id,admittime,dischtime,deathtime,admission_type,admit_provider_id,admission_locat
```

admission_type takes 6, admission_location takes 8, insurance takes 10, and ethnicity takes 13.

Regarding the count for each unique value of these variables For admission_type:

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

```
9
```

For admission_location:

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

```
12
```

For insurance:

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

```
6
```

For ethnicity:

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

```
33
```

In decreasing order, the count of unique values are 33, 12, 9, 6.

7. The `icustays.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:**

```
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
```

Count ICU stays:

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

```
94458
```

Count unique patients:

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

```
65366
```

Therefore, there are 94458 ICU stays and 65366 unique patients.

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: First, we unzip the labevent file:

```
gzip -dk ~/mimic/hosp/labevents.csv.gz
```

The size of the compressed and uncompressed file respectively:

```
ls -lh ~/mimic/hosp/labevents.csv.gz
ls -lh ~/mimic/hosp/labevents.csv
```

```
-rwxrwxrwx  1  zxhyu  zxhyu  2.5G  Jan  21  23:41  /home/zxhyu/mimic/hosp/
labevents.csv.gz
-rwxrwxrwx 1 zxhyu zxhyu 18G Jan 21 23:41 /home/zxhyu/mimic/hosp/labevents.csv
```

The run time of the compressed and uncompressed files respectively:

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

```
158374765

real    2m25.752s
user    1m39.220s
sys 0m18.850s
158374765 /home/zxhyu/mimic/hosp/labevents.csv

real    6m1.869s
user    0m0.988s
sys 0m7.183s
```

Regarding trade offs between storage and speed for big data files, it is obvious from my result that the compressed file takes up significantly less storage space than the uncompressed file, and the compressed file has shorter run times than the uncompressed file. While expectively the uncompressed file would take shorter time to run than the compressed file since the system could read the data directly without spending time to unzip the file, but the reason for the longer processing time for the uncompressed file in my case could be due to the large size of the file.

Delete the large decompressed labevent file:

```
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
```

Solution: Done.

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.

```
for char in Elizabeth Jane Lydia Darcy
do
  echo $char:
  grep -o "\b$char\b" pg42671.txt | wc -l
done
```

```
Elizabeth:
634
Jane:
293
Lydia:
170
Darcy:
416
```

Solution: `wget -nc` prevents the command from overwriting an existing file with the same name in the current directory; Elizabeth was mentioned 634 times, Jane was mentioned 293 times, Lydia was mentioned 170 times, and Darcy was mentioned 416 times.

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

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

and

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

Solution: Although both commands create the file if the file doesn't exist, the first command overwrites the file if it already exists and delete any existing content in test1.txt while the second command doesn't overwrite the file even if it already exists but instead preserves the existing content and adds the new text at the end.

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

```
chmod +x middle.sh
./middle.sh pg42671.txt 20 5
```

```


Release date: May 9, 2013 [eBook #42671]

Language: English
```

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: The above is my output. I have these outputs because the head command takes the first 20 lines (-n "$2") from the file pg42671.txt ("$1") and the tail command takes the last 5 lines (-n "$3") from the output of the head command. This results in the 5 lines from line 16 to line 20 of the original file and my outputs are part of lines 16 to 20 in pg42671.txt.

`"$1"` stands for the first argument passed to the script (pg42671.txt in this case) and this is the name of the file to be processed. `"$2"` stands for the second argument (20), which specifies the number of lines to select using the head command. `"$3"` stands for the third argument (5), which specifies the number of lines to select from the output of the head command using tail.

We need the first line of the shell script because it specifies the interpreter to be used to execute the script and preventing the system to try to interpret the script with a different shell.

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
```

For `cal`, it displays the calendar for the 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                     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
```

For `cal 2025`, it displays the calendar for the entire year 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
```

For `cal 9 1752`, it displays the calendar for September 1752. However, the calender is missing dates from September 3 to 13;

```
date
```

```
Fri Jan 24 01:20:13 PST 2025
```

For `date`, it displays the current date and time;

```
hostname
```

```
Zhiyuan
```

For `hostname`, it displays the name of my computer or system (Zhiyuan);

```
arch
```

```
x86_64
```

For `arch`, it shows my system's architecture(x86_64);

```
uname -a
```

```
Linux Zhiyuan 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
```

For `uname -a`, it displays detailed information about my laptop's system, including the kernel name, version, and system architecture;

```
uptime
```

```
 01:20:13 up  6:58,  1 user,  load average: 1.00, 0.99, 0.84
```

For `uptime`, it displays the current time, how long the system has been running, the number of users(for me there is just one), and the system load average;

```
whoami
```

```
zxhyu
```

For `who am i`, it displays information about my current session on who logged in the system;

```
who
```

```
zxhyu    pts/1        2025-01-23 18:04
```

For `who`, it displays information about the user logged in, terminal, and login time;

```
w
```

```
 01:20:13 up  6:58,  1 user,  load average: 1.00, 0.99, 0.84
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
zxhyu    pts/1    -                18:04    7:15m  0.00s   ?    -bash
```

For `w`, its display is more like a combination of the display by running `uptime`, `who am i`, and `who` and some additional information;

```
id
```

```
uid=1000(zxhyu) gid=1000(zxhyu) groups=1000(zxhyu),4(adm),20(dialout),24(cdrom),25(floppy),27(sudo
```

For `id`, it displays my ID (UID), group ID (GID), and group memberships for the current user;

```
last | head
```

```
reboot   system boot  5.15.167.4-micro Thu Jan 23 18:04   still running
reboot   system boot  5.15.167.4-micro Thu Jan 23 15:05   still running
reboot   system boot  5.15.167.4-micro Thu Jan 23 13:27   still running
reboot   system boot  5.15.167.4-micro Thu Jan 23 11:58   still running
reboot   system boot  5.15.167.4-micro Wed Jan 22 18:14   still running
reboot   system boot  5.15.167.4-micro Wed Jan 22 17:55   still running
reboot   system boot  5.15.167.4-micro Tue Jan 21 23:41   still running
reboot   system boot  5.15.167.4-micro Tue Jan 21 14:56   still running
reboot   system boot  5.15.167.4-micro Tue Jan 21 11:35   still running
reboot   system boot  5.15.167.4-micro Mon Jan 20 18:42   still running
```

For `last | head`, it displays first 10 most recent login history of users due to head;

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

```
consents consented confers confered presents presented prefers prefered
```

For `echo {con,pre}{sent,fer}{s,ed}`, it shows all possible combinations of the text inside the curly braces ({}), for me they are consents consented confers confered presents presented prefers prefered;

```
time sleep 5
```

```

real    0m5.003s
user    0m0.001s
sys 0m0.000s
```

For `time sleep 5`, it measures the time it takes to execute the sleep 5 command, which pauses the terminal for 5 seconds. For me it shows real(0m5.004s), user(0m0.002s), and sys time(0m0,000s);

```
history | tail
```

For `history | tail`, it displays the most recent 10 commands I executed from my bash command history.

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:** Here is the screenshot of Section 4.1.5:

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.