Biostat 203B Homework 1

Due Jan 24, 2025 @ 11:59PM

Sakshi Oza, 606542442

Display machine information for reproducibility:

```r
sessionInfo()
```

```
R version 4.4.1 (2024-06-14)
Platform: aarch64-apple-darwin20
Running under: macOS Sonoma 14.4

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.4-arm64/Resources/lib/libRlapack.dylib;

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_4.4.1    fastmap_1.2.0     cli_3.6.3         tools_4.4.1
 [5] htmltools_0.5.8.1 rstudioapi_0.17.0 yaml_2.3.10       rmarkdown_2.29
 [9] knitr_1.48        jsonlite_1.8.9    xfun_0.49         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 1

Q1 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 2

Here is my link for completion certificate and completion report

Q3. Linux Shell Commands

Solution 3.1

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 56
-rw-rw-r--@  1 sakshihiteshoza  staff  15199 Oct 10 13:29 CHANGELOG.txt
-rw-rw-r--@  1 sakshihiteshoza  staff   2518 Oct 10 14:30 LICENSE.txt
-rw-rw-r--@  1 sakshihiteshoza  staff   2884 Oct 11 14:55 SHA256SUMS.txt
drwxr-xr-x@ 25 sakshihiteshoza  staff    800 Jan 24 16:36 hosp
drwxr-xr-x@ 12 sakshihiteshoza  staff    384 Jan 21 16:19 icu
-rw-rw-r--@  1 sakshihiteshoza  staff    789 Dec 28 18:04 index.html
```

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.

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 3.2

The primary reason for using .csv.gz over .csv is to improve the efficiency of file storage and transfer, especially for large datasets, without losing any data quality or integrity.Many tools and programming languages (like Python, R, and Unix-based systems) can automatically decompress .gz files without needing additional steps from the user. This makes it convenient for those processing the data, as they can directly load and work with compressed data.

```
# Solution 3.2
ls -l ~/mimic/hosp
ls -l ~/mimic/icu
```

```
total 12306256
-rw-rw-r--@ 1 sakshihiteshoza  staff    19928140 Jun 24  2024 admissions.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff      427554 Apr 12  2024 d_hcpcs.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff      876360 Apr 12  2024 d_icd_diagnoses.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff      589186 Apr 12  2024 d_icd_procedures.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff       13169 Oct  3 06:07 d_labitems.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff    33564802 Oct  3 06:07 diagnoses_icd.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff     9743908 Oct  3 06:07 drgcodes.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff   811305629 Apr 12  2024 emar.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff   748158322 Apr 12  2024 emar_detail.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff     2162335 Apr 12  2024 hcpcsevents.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff        2907 Dec 28 18:04 index.html
-rw-r--r--@ 1 sakshihiteshoza  staff  2592909134 Jan 24 15:14 labevents.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff   117644075 Oct  3 06:08 microbiologyevents.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff    44069351 Oct  3 06:08 omr.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff     2835586 Apr 12  2024 patients.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff   525708076 Apr 12  2024 pharmacy.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff   666594177 Apr 12  2024 poe.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff    55267894 Apr 12  2024 poe_detail.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff   606298611 Apr 12  2024 prescriptions.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff     7777324 Apr 12  2024 procedures_icd.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff      127330 Apr 12  2024 provider.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff     8569241 Apr 12  2024 services.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff    46185771 Oct  3 06:08 transfers.csv.gz
total 8506792
-rw-rw-r--@ 1 sakshihiteshoza  staff       41566 Apr 12  2024 caregiver.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff  3502392765 Apr 12  2024 chartevents.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff       58741 Apr 12  2024 d_items.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff    63481196 Apr 12  2024 datetimeevents.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff     3342355 Oct  3 04:36 icustays.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff        1336 Dec 28 18:04 index.html
-rw-rw-r--@ 1 sakshihiteshoza  staff   311642048 Apr 12  2024 ingredientevents.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff   401088206 Apr 12  2024 inputevents.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff    49307639 Apr 12  2024 outputevents.csv.gz
-rw-rw-r--@ 1 sakshihiteshoza  staff    24096834 Apr 12  2024 procedureevents.csv.gz
```

Solution 3.3

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

a. `zcat` - Displays the contents of compressed `.gz` files without uncompressing them. Example usage:

```
# The output is very long and so commenting the code.
# zcat < ~/mimic/hosp/poe.csv.gz
```

b. `zless` - Opens compressed `.gz` files for reading, similar to the less command.

```
# zless  ~/mimic/hosp/poe.csv.gz
```

c.`zmore`- Similar to zless, but works like more to display the contents of compressed .gz files page by page.

```
# zmore  ~/mimic/hosp/poe.csv.gz
```

d.`zgrep`- Searches for a pattern in compressed .gz files, similar to grep.

```
# zgrep "pattern" ~/mimic/hosp/poe.csv.gz
```

Solution 3.4

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
```

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 "Number of lines in" $datafile "is"
  zcat < $datafile | wc -l
done
```

```
Number of lines in /Users/sakshihiteshoza/mimic/hosp/admissions.csv.gz is
  546029
Number of lines in /Users/sakshihiteshoza/mimic/hosp/labevents.csv.gz is
 158374765
Number of lines in /Users/sakshihiteshoza/mimic/hosp/patients.csv.gz is
  364628
```

Solution 3.5

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.)

```
# displaying first few lines of the file
zcat < ~/mimic/hosp/admissions.csv.gz | head
```

```
subject_id,hadm_id,admittime,dischtime,deathtime,admission_type,admit_provider_id,admission_l
10000032,22595853,2180-05-06 22:23:00,2180-05-07 17:15:00,,URGENT,P49AFC,TRANSFER FROM HOSPIT
10000032,22841357,2180-06-26 18:27:00,2180-06-27 18:49:00,,EW EMER.,P784FA,EMERGENCY ROOM,HOM
10000032,25742920,2180-08-05 23:44:00,2180-08-07 17:50:00,,EW EMER.,P19UTS,EMERGENCY ROOM,HOS
10000032,29079034,2180-07-23 12:35:00,2180-07-25 17:55:00,,EW EMER.,P06OTX,EMERGENCY ROOM,HOM
10000068,25022803,2160-03-03 23:16:00,2160-03-04 06:26:00,,EU OBSERVATION,P39NWO,EMERGENCY RO
10000084,23052089,2160-11-21 01:56:00,2160-11-25 14:52:00,,EW EMER.,P42H7G,WALK-IN/SELF REFER
10000084,29888819,2160-12-28 05:11:00,2160-12-28 16:07:00,,EU OBSERVATION,P35NE4,PHYSICIAN RE
10000108,27250926,2163-09-27 23:17:00,2163-09-28 09:04:00,,EU OBSERVATION,P40JML,EMERGENCY RO
10000117,22927623,2181-11-15 02:05:00,2181-11-15 14:52:00,,EU OBSERVATION,P47EY8,EMERGENCY RO
```

```
# Number of rows in the datafile excluding headers
zcat < ~/mimic/hosp/admissions.csv.gz  | tail -n +2 | wc -l
```

```
  546028
```

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

```
  546028
```

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

```
  223452
```

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

```
364627
```

The difference in the number of unique patients between the two files suggests that not all patients in the patients.csv.gz file are represented in the admissions.csv.gz file. This could indicate that some patients were not included in the admissions.csv.gz file.

Solution 3.6

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.)

```
# Possible values taken by admission type
zcat < ~/mimic/hosp/admissions.csv.gz | tail -n +2 | awk -F',' '{print $6}' |
sort | uniq -c | sort -nr
```

```
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
```

```
# Possible values taken by admission_location
zcat < ~/mimic/hosp/admissions.csv.gz | tail -n +2 | awk -F',' '{print $8}' |
sort | uniq -c | sort -nr
```

```
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
```

```
# Possible values taken by insurance
zcat < ~/mimic/hosp/admissions.csv.gz | tail -n +2 | awk -F',' '{print $10}' |
sort | uniq -c | sort -nr
```

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

```
# Possible values taken by ethinicity
zcat < ~/mimic/hosp/admissions.csv.gz | tail -n +2 | awk -F',' '{print $13}' |
sort | uniq -c | sort -nr
```

```
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
```

Solution 3.7

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?

```
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 (MI
10000690,25860671,37081114,Medical Intensive Care Unit (MICU),Medical Intensive Care Unit (MI
10000980,26913865,39765666,Medical Intensive Care Unit (MICU),Medical Intensive Care Unit (MI
10001217,24597018,37067082,Surgical Intensive Care Unit (SICU),Surgical Intensive Care Unit
10001217,27703517,34592300,Surgical Intensive Care Unit (SICU),Surgical Intensive Care Unit
10001725,25563031,31205490,Medical/Surgical Intensive Care Unit (MICU/SICU),Medical/Surgical
10001843,26133978,39698942,Medical/Surgical Intensive Care Unit (MICU/SICU),Medical/Surgical
10001884,26184834,37510196,Medical Intensive Care Unit (MICU),Medical Intensive Care Unit (MI
10002013,23581541,39060235,Cardiac Vascular Intensive Care Unit (CVICU),Cardiac Vascular Inte
```

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

```
   94458
```

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

```
   65366
```

Solution 3.8

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.)

```
# Measure storage of compressed file
ls -l ~/mimic/hosp/labevents.csv.gz
```

```
-rw-r--r--@ 1 sakshihiteshoza  staff  2592909134 Jan 24 15:14 /Users/sakshihiteshoza/mimic/ho
```

```
# Measure time of compressed file
time zcat < ~/mimic/hosp/labevents.csv.gz | wc -l
```

```
 158374765

real    0m19.213s
user    0m29.826s
sys 0m2.182s
```

```
# Decompress the file
gzip -dk ~/mimic/hosp/labevents.csv.gz
```

```
# Measure storage of uncompressed file
ls -l ~/mimic/hosp/labevents.csv
```

```
-rw-r--r--  1 sakshihiteshoza  staff  18402851720 Jan 24 15:14 /Users/sakshihiteshoza/mimic/l
```

```
# Measure time of uncompressed file
time wc -l ~/mimic/hosp/labevents.csv
```

```
 158374765 /Users/sakshihiteshoza/mimic/hosp/labevents.csv

real    0m18.502s
user    0m16.314s
sys 0m1.511s
```

```
# Delete the uncompressed file
rm ~/mimic/hosp/labevents.csv
```

Explanation

1. **Storage Comparison:**
   - **Compressed File:** The compressed file, `labevents.csv.gz`, has a size of **2.59 GB** (2,592,909,134 bytes). This is a significant reduction in size compared to the uncompressed version, making it much more storage-efficient. Compression is beneficial when dealing with large datasets, as it helps conserve disk space, which can be critical in large-scale data projects.
   - **Uncompressed File:** After decompression, the file size grows to a massive **18.4 GB** (18,402,851,720 bytes). This is typical for CSV files, which often contain large amounts of raw data. While the uncompressed file is much larger, it is easier and faster to process since there is no need to decompress it first.
2. **Time Comparison:**
   - **Compressed File (`zcat`):** The time to process the compressed file with the command `zcat < labevents.csv.gz | wc -l` took approximately **19.3 seconds (real time)**. The high **user time of 30.07 seconds** reflects the extra computational cost associated with decompressing the file on the fly, which slows down the operation compared to uncompressed data.
   - **Uncompressed File (`wc -l`):** Processing the uncompressed file with `wc -l` took about **18.6 seconds (real time)**, which is slightly faster than the compressed version. This is because the file is directly read from disk without the overhead of decompression. The **user time is much lower at 16.43 seconds**, indicating that less CPU power was needed to process the data compared to the compressed version.

Q4. Who’s popular in Price and Prejudice

Solution 4.1

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.

```
# Add wget PATH on mac
PATH=$PATH:/opt/homebrew/bin
wget -nc http://www.gutenberg.org/cache/epub/42671/pg42671.txt
```

```
File ‘pg42671.txt’ already there; not retrieving.
```

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 -i $char pg42671.txt | wc -l
done
```

```
Elizabeth:
     634
Jane:
     293
Lydia:
     171
Darcy:
     418
```

Explanation

`wget -nc` is a command used to download files from the web using the wget tool with the -nc option, which stands for “no clobber.” `wget`: A command-line tool used to download files from the internet. It supports downloading files over HTTP, HTTPS, and FTP protocols. `-nc` (no clobber): Prevents `wget` from overwriting an existing file. If the file we’re trying to download already exists in the directory, the -nc option ensures that `wget` does not re-download the file or overwrite it. - If the file does not exist: wget will download the file normally. - If the file already exists: wget will skip downloading the file and leave the existing file unchanged. - If partial download exists: wget -nc will not continue or restart an incomplete file download. According to this **Elizabeth** is repeated the most times (634).

Solution 4.2

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

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

and

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

Explanation

a. `echo 'hello, world' > test1.txt`: The `>` operator is used for output redirection. This command writes `'hello, world'` to the file `test1.txt`. If the file already exists, it will be overwritten.

b. `echo 'hello, world' >> test2.txt`: The `>>` operator is used for append redirection. This command appends `'hello, world'` to the file `test2.txt`. If the file already exists, the new text will be added to the end of the file without deleting the existing content. If the file doesn't exist, it will create the file and then write `'hello, world'` to it.

Solution 4.3

3. Using your favorite text editor (e.g., `vi`), type the following and save the file as `middle.sh`: Using `chmod` to make the file executable by the owner, and run

```
#!/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"
```

```
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?

Explanation

output : head -n 20 pg42671.txt: This command extracts the first 20 lines of pg42671.txt. tail -n 5: From the 20 lines output by head, this command takes the last 5 lines.

The first line of the script,`#!/bin/sh`, is known as the shebang. -It tells the operating system which interpreter to use for running the script. In this case, the shell interpreter located at `/bin/sh`is used. -Without this line, the system might not know how to execute the script, especially if we're running it directly from the command line.

In shell scripts, `"$1"`, `"$2"`, and `"$3"` are positional parameters that correspond to the arguments passed when we run the script: -`"$1"`: The first argument passed to the script. In this case, it's the filename (`pg42671.txt`). -`"$2"`: The second argument, specifying the number of lines to extract from the start of the file (used by the head command). In this case, 20. -`"$3"`: The third argument, specifying the number of lines to extract from the bottom of the result (used by the tail command). In this case, 5.

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 5

a. **`cal`**:
   This command shows us the current month's calendar. For example, if it's January, it'll display the calendar for January.

```
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
```

b. **`cal 2025`**:
This will display the entire calendar for the year 2025. It's just a way to see what days of the week certain months fall on.

```
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
```

c. **`cal 9 1752`**:
This one shows the calendar for September 1752. The `cal 9 1752` command is special because September 1752 had a unique situation where 11 days were skipped when the Gregorian calendar was adopted, so we'll see gaps in the calendar.

```
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
```

d. **`date`**:
It shows the current date and time on our system.

```
date
```

```
Fri Jan 24 16:48:26 PST 2025
```

e. **`hostname`**:
This gives the name of my computer or server.

```
hostname
```

```
Sakshis-MacBook-Air.local
```

f. **`arch`**:
Tells us what type of architecture our system is using (like whether it's a 64-bit or 32-bit computer).

```
arch
```

```
arm64
```

g. **`uname -a`**:
This one gives us a whole bunch of details about our operating system, like what version of Linux you're running, the kernel version, and more.

```
uname -a
```

```
Darwin Sakshis-MacBook-Air.local 23.4.0 Darwin Kernel Version 23.4.0: Wed Feb 21 21:51:37 PS
```

h. **`uptime`**:
Tells us how long our computer has been running without being rebooted, along with the current time and a little information on how busy our system is right now.

```
uptime
```

```
16:48  up  5:43, 2 users, load averages: 2.32 1.89 1.82
```

i. **`who am i`**:
This shows who we are , where we're logged in from, and when we logged in.

```
who am i
```

```
sakshihiteshoza                Jan 24 16:48
```

j. **`who`**:
Lists all the people currently logged into our system, along with where we are logged in from and when we logged in.

```
who
```

```
sakshihiteshoza  ttys001      Jan 24 15:14
sakshihiteshoza  console      Jan 24 14:46
```

k. **`w`**:
Similar to `who`, but it gives a bit more detail. It shows who's logged in, what they're doing, how long they've been idle, and the system load.

```
w
```

```
16:48  up  5:43, 2 users, load averages: 2.32 1.89 1.82
USER       TTY      FROM    LOGIN@  IDLE WHAT
sakshihite s001     -      15:14    1:33 -zsh
sakshihite console  -      14:46    2:01 -
```

l. **id**:
   Displays your user ID (UID), group ID (GID), and the groups you're part of.

```
id
```

```
uid=501(sakshihiteshoza) gid=20(staff) groups=20(staff),12(everyone),61(localaccounts),79(_ap
```

m. **last | head**:
   This shows the last few logins to the system (using the `last` command). It shows you who's logged in, when they logged in, and where from, but with the `head` command, it limits it to the most recent 10.

```
last | head
```

```
sakshihiteshoza ttys001                         Fri Jan 24 15:14   still logged in
sakshihiteshoza console                         Fri Jan 24 14:46   still logged in
sakshihiteshoza ttys001                         Fri Jan 24 12:25 - 12:25  (00:00)
sakshihiteshoza ttys001                         Fri Jan 24 12:19 - 12:19  (00:00)
sakshihiteshoza ttys001                         Fri Jan 24 12:03 - 12:03  (00:00)
sakshihiteshoza ttys001                         Fri Jan 24 10:38 - 10:38  (00:00)
sakshihiteshoza console                         Fri Jan 24 10:01 - 14:46  (04:45)
reboot time                            Fri Jan 24 10:01
sakshihiteshoza ttys000                         Thu Jan 23 17:36 - 17:36  (00:00)
sakshihiteshoza ttys001                         Thu Jan 23 17:25 - 17:25  (00:00)
```

n. **echo {con,pre}{sent,fer}{s,ed}**:
   This is a called "brace expansion" that combines different parts of words . It will create a list like:

   - cons
   - sent
   - s
   - ed
   - present
   - fer

- s
- ed

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

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

o. **`time sleep 5`**:
This command measures how long the `sleep 5` command takes to run. Since `sleep 5` just makes the system pause for 5 seconds, it will show you that it took exactly 5 seconds to run.

```
time sleep 5
```

```


real    0m5.012s
user    0m0.000s
sys 0m0.001s
```

p. **`history | tail`**:
Shows you the last few commands you've typed in your terminal. If we have been working on a project, it's like looking back at our "command history" to see what we did recently.

```
history | tail
```

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`.

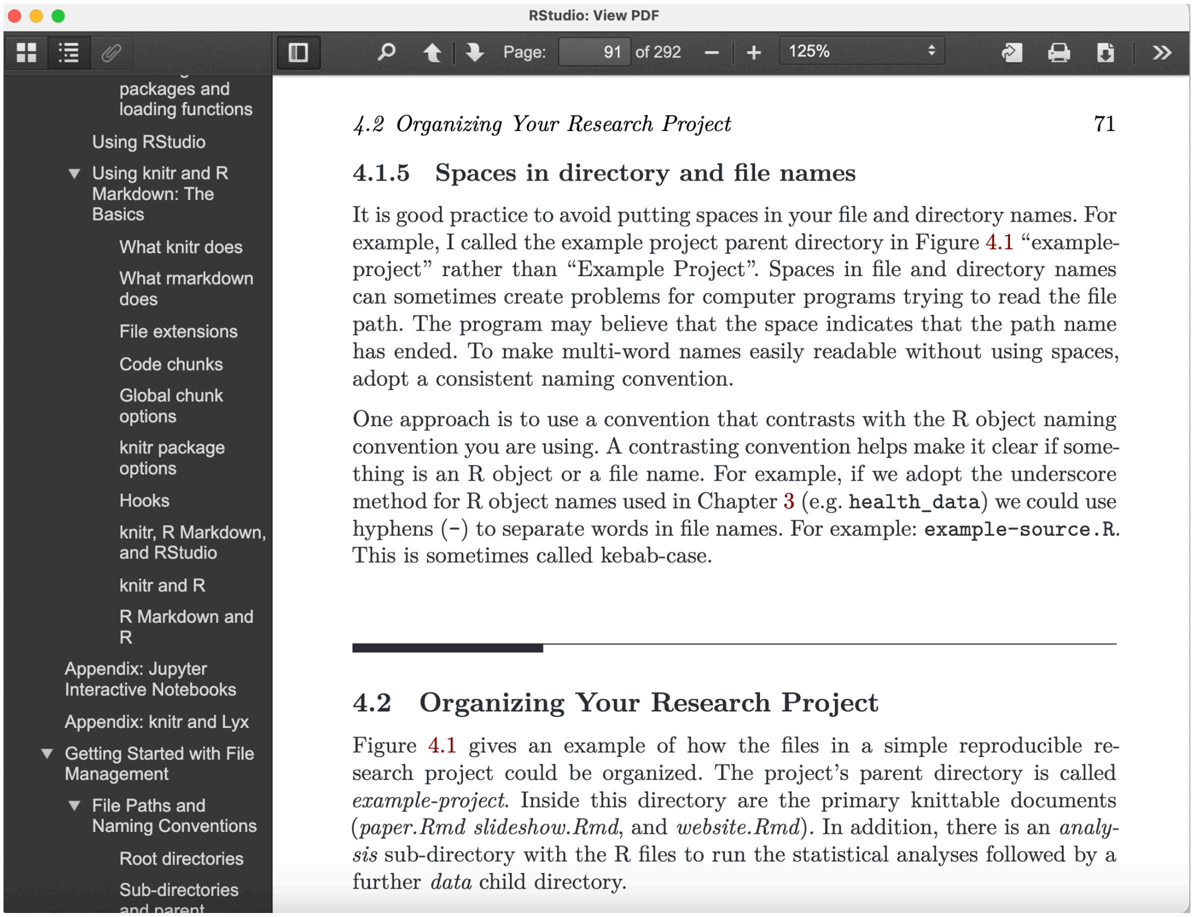
2. Open the project by clicking `rep-res-3rd-edition.Rproj` and compile the book by clicking `Build Book` in the `Build` panel of RStudio. (Hint: I was able to build `git_book` and `epub_book` directly. For `pdf_book`, I needed to add a line `\usepackage{hyperref}` to the file `Rep-Res-Book/rep-res-3rd-edition/latex/preabmle.tex`.)

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 6

a. PDF build


RStudio: View PDF
Page: 91 of 292
125%
packages and loading functions
Using RStudio
Using knitr and R Markdown: The Basics
What knitr does
What rmarkdown does
File extensions
Code chunks
Global chunk options
knitr package options
Hooks
knitr, R Markdown, and RStudio
knitr and R
R Markdown and R
Appendix: Jupyter Interactive Notebooks
Appendix: knitr and Lyx
Getting Started with File Management
File Paths and Naming Conventions
Root directories
Sub-directories and parent

4.2 Organizing Your Research Project 71

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.

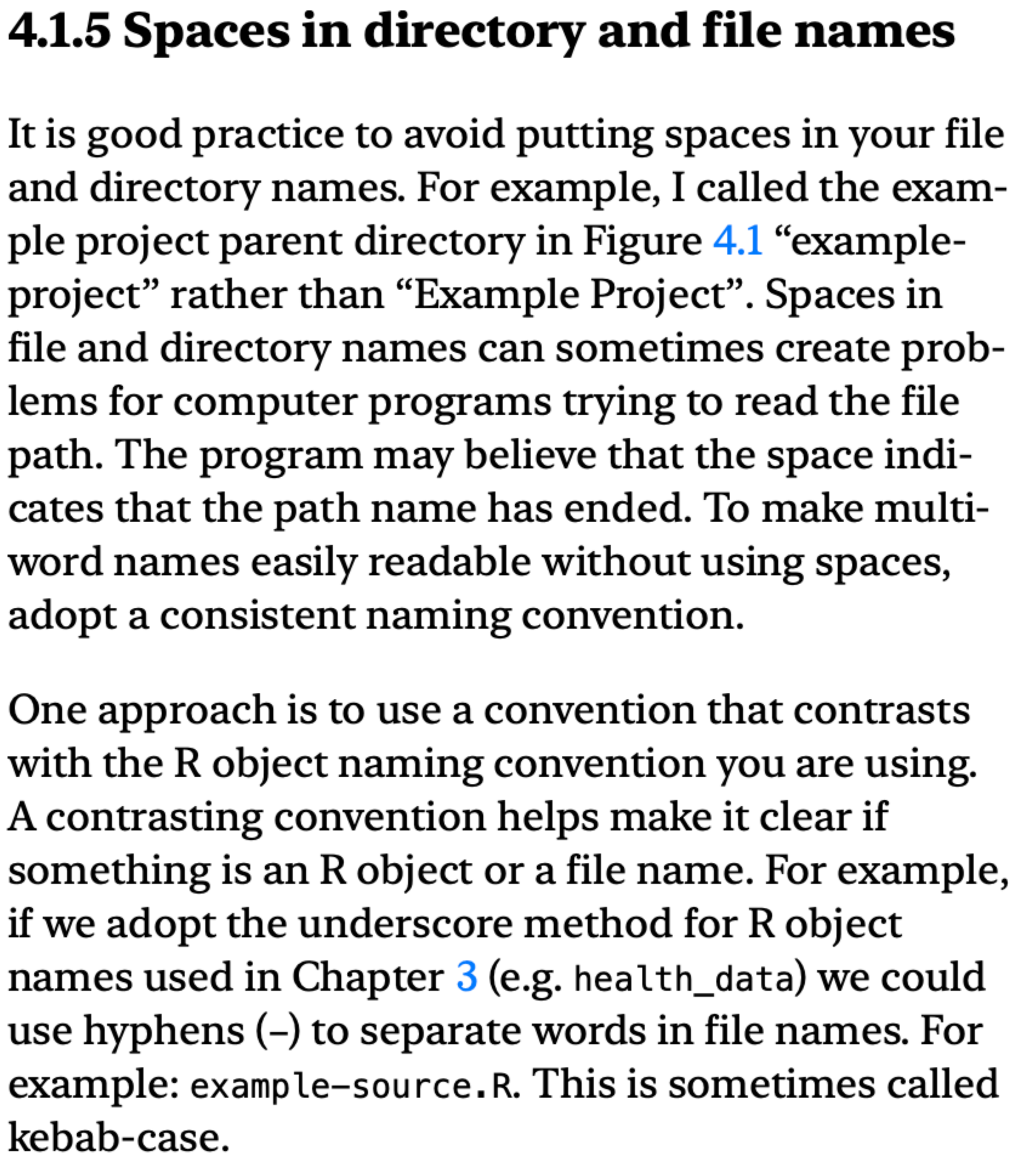
One approach is to use a convention that contrasts with the R object naming convention you are using. A contrasting convention helps make it clear if something is an R object or a file name. For example, if we adopt the underscore method for R object names used in Chapter 3 (e.g. `health_data`) we could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

Figure 1: PDF

b. GitHub build



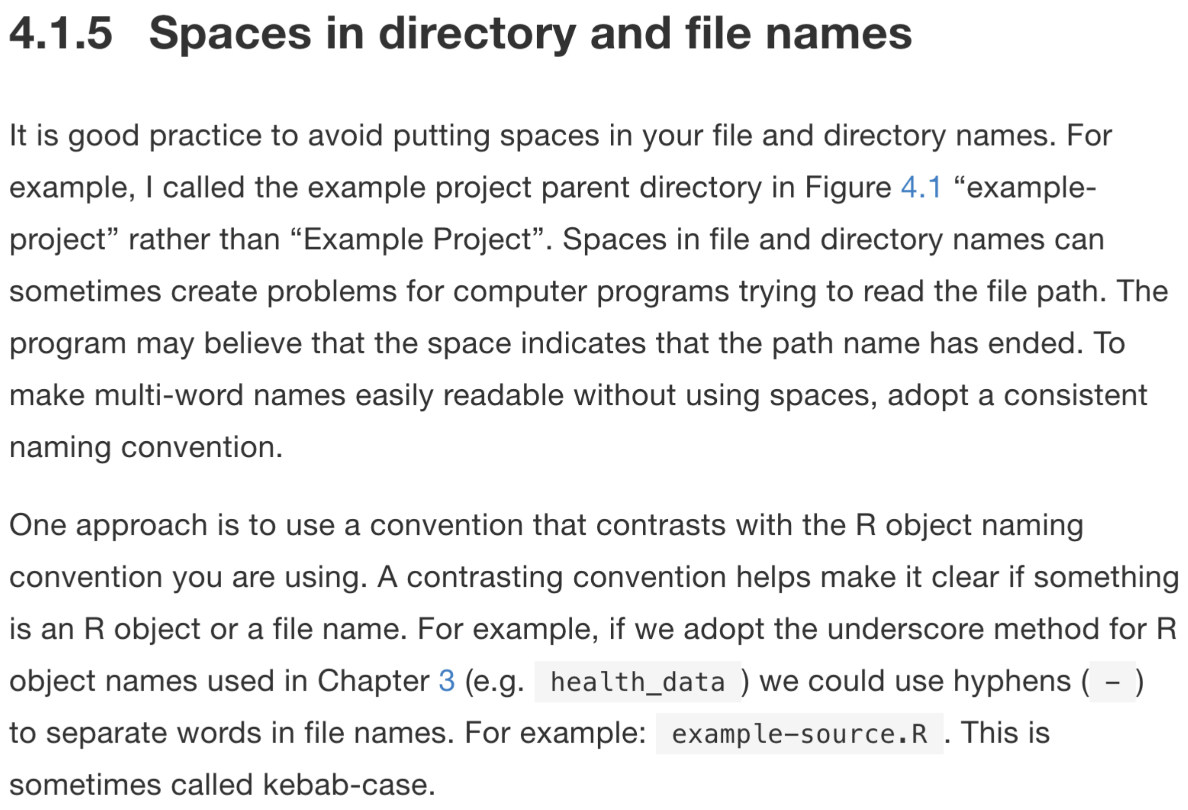
4.1.5 Spaces in directory and file names

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.

Figure 2: Github

c. EPUB build



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.

Figure 3: EPUB