Assignment 1: Manipulation of mammalian sleep data

Your name and student ID

Today's date

```
BEGIN ASSIGNMENT
requirements: requirements.R
generate: true
files:
  - data
```

Instructions

- Solutions will be released on Tuesday, January 26
- This semester, homework assignments are for practice only and will not be turned in for marks.

Helpful hints:

- Every function you need to use was taught during lecture! So you may need to revisit the lecture code to help you along by opening the relevant files on Datahub. Alternatively, you may wish to view the code in the condensed PDFs posted on the course website. Good luck!
- Knit your file early and often to minimize knitting errors! If you copy and paste code for the slides, you are bound to get an error that is hard to diagnose. Typing out the code is the way to smooth knitting! We recommend knitting your file each time after you write a few sentences/add a new code chunk, so you can detect the source of knitting errors more easily. This will save you and the GSIs from frustration!
- It is good practice to not allow your code to run off the page. To avoid this, have a look at your knitted PDF and ensure all the code fits in the file. If it doesn't look right, go back to your .Rmd file and add spaces (new lines) using the return or enter key so that the code runs onto the next line.

Begin by knitting this document by pushing the "Knit" button above. As you fill in code and text in the document, you can re-knit (push the button again) and see how the document changes. It is important to re-knit often, because if there is any error in your code, the file will not generate a PDF, so our advice is to knit early and often!

Using `dplyr` to investigate sleep times in mammals

The data file `sleep.csv` contains the sleeptimes and weights for a set of mammals. Hit the green arrow icon in the line below to execute the two lines of code in the code chunk, or execute them line by line by placing your cursor on the first line and hitting cmd + enter on Mac or ctrl + enter on PC.

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following object is masked from 'package:testthat':
##
##     matches

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(readr)
sleep <- read_csv("data/sleep.csv")
```

```
## Parsed with column specification:
## cols(
##   name = col_character(),
##   genus = col_character(),
##   vore = col_character(),
##   order = col_character(),
##   conservation = col_character(),
##   sleep_total = col_double(),
##   sleep_rem = col_double(),
##   sleep_cycle = col_double(),
##   awake = col_double(),
##   brainwt = col_double(),
##   bodywt = col_double()
## )
```

- The `library` command loads the library `dplyr` into memory.
- The `readr` library contains functions to read in the dataset.
- The `dplyr` library contains functions we will use to manipulate data.

Notice that an object called `sleep` appeared in the Environment tab under "Data".

1. [2 points] Use four useful functions discussed in lecture to examine the sleep data set:

```r
# Text inside a code chunk that begins with "#" is called a comment.
# We sometimes use comments to explain code to you in plain English.
# Write your four functions below these comments, replacing the placeholder
# text "<<<<YOUR CODE HERE>>>>". Remember, code does *not* begin with a "#"

"<<<<YOUR CODE HERE>>>>"
```

```
## [1] "<<<<YOUR CODE HERE>>>>"
```

```r
"<<<<YOUR CODE HERE>>>>"
```

```
## [1] "<<<<YOUR CODE HERE>>>>"
```

```r
"<<<<YOUR CODE HERE>>>>"
```

```
## [1] "<<<<YOUR CODE HERE>>>>"
```

```r
"<<<<YOUR CODE HERE>>>>"
```

```
## [1] "<<<<YOUR CODE HERE>>>>"
```

```r
# Then, assign p1 to a vector of your function names, in alphabetical order.
# For example, assigning p0 to a vector of fruits looks like this:
# p0 <- c("apple", "banana", "orange")
```

```r
p1 <- c("dim", "head", "names", "str") #SOLUTION
```

```
BEGIN QUESTION
name: p1
manual: false
points: 2
```

```r
## Test ##
test_that("p1a", {
  expect_true(length(p1) == 4)
  print("p1a: Checking p1 has 4 items in a list")
})
```

```
## [1] "p1a: Checking p1 has 4 items in a list"
```

```r
## Test ##
test_that("p1b", {
           expect_true(p1[1] == "dim" & p1[2] == "head" & p1[3] == "names" & p1[4] == "str" )
   print("p1b: Checking the names of the 4 functions in alphabetical order")
})
```

```
## [1] "p1b: Checking the names of the 4 functions in alphabetical order"
```

Description of the variables found in the sleep dataset:

Column name	Description
name	common name
genus	taxonomic rank
vore	carnivore, omnivore or herbivore?
order	taxonomic rank
conservation	the conservation status of the mammal
sleep_total	total amount of sleep, in hours
sleep_rem	Rapid eye movement (REM) sleep, in hours
sleep_cycle	length of sleep cycle, in hours
awake	amount of time spent awake, in hours
brainwt	brain weight in kilograms
bodywt	body weight in kilograms

2. [2 points] Write code to select a set of columns. Specifically select the `awake`, `brainwt`, and `bodywt` columns. Assign this smaller dataset to a data frame called `sleep_small`

```
sleep_small <- select(sleep, awake, brainwt, bodywt) #SOLUTION
```

```
BEGIN QUESTION
name: p2
manual: false
points: 2
```

```
## Test ##
test_that("p2a", {
  expect_true(is.data.frame(sleep_small))
  print("p2a: Checking sleep_small is a dataframe")
})
```

```
## [1] "p2a: Checking sleep_small is a dataframe"
```

```
## Test ##
test_that("p2b", {
           expect_true(ncol(sleep_small) == 3)
   print("p2b: Checking sleep_small has 3 columns")
})
```

```
## [1] "p2b: Checking sleep_small has 3 columns"
```

```
## Test ##
test_that("p2c", {
           expect_true(all(names(sleep_small) == c("awake", "brainwt", "bodywt")))
   print("p2c: Checking sleep_small has the 'awake', 'brainwt', and 'bodywt'")
})
```

```
## [1] "p2c: Checking sleep_small has the 'awake', 'brainwt', and 'bodywt'"
```

3. [1 point] To select a range of columns by name, use the ":" (colon) operator. Redo the selection for question 1, but use the colon operator. Assign this to `sleep_small_colon`. Note that this returns the same data frame as the previous problem, but is not recommended in practice because it depends on the ordering of the columns and isn't explicit in the columns that are selected, whereas selection by name offers much higher readability for someone else looking at your code later on.

```r
sleep_small_colon <- sleep %>% select(awake:bodywt) #SOLUTION
```

```
BEGIN QUESTION
name: p3
manual: false
points: 1
```

```r
## Test ##
test_that("p3a", {
  expect_true(is.data.frame(sleep_small_colon))
  print("p3a: Checking sleep_small_colon is a dataframe")
})
```

```
## [1] "p3a: Checking sleep_small_colon is a dataframe"
```

```r
## Test ##
test_that("p3b", {
           expect_true(ncol(sleep_small_colon) == 3)
   print("p3b: Checking sleep_small_colon has 3 columns")
})
```

```
## [1] "p3b: Checking sleep_small_colon has 3 columns"
```

```r
## Test ##
test_that("p3c", {
           expect_true(all(names(sleep_small_colon) == c("awake", "brainwt", "bodywt")))
   print("p3c: Checking sleep_small_colon has the 'awake', 'brainwt', and 'bodywt'")
})
```

```
## [1] "p3c: Checking sleep_small_colon has the 'awake', 'brainwt', and 'bodywt'"
```

4. [1 point] From the original dataset `sleep` select all the columns except for the `vore` variable. Assign this to `sleep_no_vore`.

```
sleep_no_vore <- sleep %>% select(-vore) #SOLUTION
```

```
BEGIN QUESTION
name: p4
manual: false
points: 1
```

```
## Test ##
test_that("p4a", {
  expect_true(is.data.frame(sleep_no_vore))
  print("p4a: Checking sleep_small_colon is a dataframe")
})
```

```
## [1] "p4a: Checking sleep_small_colon is a dataframe"
```

```
## Test ##
test_that("p4b", {
           expect_true(ncol(sleep_no_vore) == 10)
   print("p4b: Checking sleep_small_colon has 3 columns")
})
```

```
## [1] "p4b: Checking sleep_small_colon has 3 columns"
```

```
## Test ##
test_that("p4c", {
           expect_true(!("vore" %in% names(sleep_no_vore)))
   print("p4c: Checking sleep_no_vore has no columns with 'vore'")
})
```

```
## [1] "p4c: Checking sleep_no_vore has no columns with 'vore'"
```

5. [1 point] Run the following chunk of code.

```r
select(sleep, starts_with("sl"))
```

```
## Warning: ‘...‘ is not empty.
##
## We detected these problematic arguments:
## * ‘needs_dots‘
##
## These dots only exist to allow future extensions and should be empty.
## Did you misspecify an argument?
```

```
## # A tibble: 83 x 3
##    sleep_total sleep_rem sleep_cycle
##          <dbl>     <dbl>       <dbl>
##  1        12.1      NA        NA
##  2        17         1.8      NA
##  3        14.4       2.4      NA
##  4        14.9       2.3       0.133
##  5         4         0.7       0.667
##  6        14.4       2.2       0.767
##  7         8.7       1.4       0.383
##  8         7        NA        NA
##  9        10.1       2.9       0.333
## 10         3        NA        NA
## # ... with 73 more rows
```

What does it return? Copy your choice and assign it to p5

```r
# p5 <- "returns the number of columns that start with sl"
# p5 <- "returns all columns that start with sl"
# p5 <- "returns all rows that start with sl"
# p5 <- "returns all animals whose names start with sl""
```

```r
p5 <- "returns all columns that start with sl" #SOLUTION
```

```
BEGIN QUESTION
name: p5
manual: false
points: 1
```

```r
## Test ##
test_that("p5", {
  expect_true(p5 == "returns all columns that start with sl")
  print("Checking response...")
})
```

```
## [1] "Checking response..."
```

```
select(sleep, starts_with("sl"))
```

```
## Warning: '...' is not empty.
##
## We detected these problematic arguments:
## * 'needs_dots'
##
## These dots only exist to allow future extensions and should be empty.
## Did you misspecify an argument?
```

```
## # A tibble: 83 x 3
##    sleep_total sleep_rem sleep_cycle
##          <dbl>     <dbl>       <dbl>
##  1        12.1      NA        NA
##  2        17         1.8      NA
##  3        14.4       2.4      NA
##  4        14.9       2.3       0.133
##  5         4         0.7       0.667
##  6        14.4       2.2       0.767
##  7         8.7       1.4       0.383
##  8         7        NA        NA
##  9        10.1       2.9       0.333
## 10         3        NA        NA
## # ... with 73 more rows
```

6. [1 point] Rewrite the previous chunk of code using the pipe operator. Assign this to sleep_sl.

```
sleep_sl <- sleep %>% select(starts_with("sl")) #SOLUTION
```

```
BEGIN QUESTION
name: p6
manual: false
points: 1
```

```
## Test ##
test_that("p6a", {
  expect_true(is.data.frame(sleep_sl))
  print("p6a: Checking sleep_sl is a dataframe.")
})
```

```
## [1] "p6a: Checking sleep_sl is a dataframe."
```

```
## Test ##
test_that("p6b", {
  expect_true(ncol(sleep_sl) == 3)
  print("p6b: Checking sleep_sl has 3 columns.")
})
```

```
## [1] "p6b: Checking sleep_sl has 3 columns."
```

```r
## Test ##
test_that("p6c", {
  expect_true(("sleep_total" %in% names(sleep_sl)) &&
                  ("sleep_rem" %in% names(sleep_sl)) &&
                  ("sleep_cycle" %in% names(sleep_sl)))
  print("p6c: Checking sleep_sl has the 3 columns that start with sl.")
})
```

```
## [1] "p6c: Checking sleep_sl has the 3 columns that start with sl."
```

7. [1 point] From the original `sleep` dataset, filter the rows for mammals that sleep a total of more than 16 hours. Assign this to `sleep_over16`.

```r
sleep_over16 <- sleep %>% filter(sleep_total > 16) #SOLUTION
```

```
BEGIN QUESTION
name: p7
manual: false
points: 1
```

```r
## Test ##
test_that("p7a", {
  expect_true(is.data.frame(sleep_over16))
  print("p7a: Checking sleep_over16 is a dataframe.")
})
```

```
## [1] "p7a: Checking sleep_over16 is a dataframe."
```

```r
## Test ##
test_that("p7b", {
  expect_true(ncol(sleep_over16) == 11)
  print("p7b: Checking sleep_over16 has 11 columns.")
})
```

```
## [1] "p7b: Checking sleep_over16 has 11 columns."
```

```r
## Test ##
test_that("p7c", {
  expect_true(nrow(sleep_over16) == 8)
  print("p7c: Checking sleep_over16 has 8 rows.")
})
```

```
## [1] "p7c: Checking sleep_over16 has 8 rows."
```

8. [2 points] Filter the rows for mammals that sleep a total of more than 16 hours and have a body weight of greater than 1 kilogram. Assign this to `sleep_mammals`.

```
sleep_mammals <- sleep %>% filter(sleep_total > 16 & bodywt > 1) #SOLUTION
```

```
BEGIN QUESTION
name: p8
manual: false
points: 2
```

```
## Test ##
test_that("p8a", {
  expect_true(is.data.frame(sleep_mammals))
  print("p8a: Checking sleep_mammals is a dataframe.")
})
```

```
## [1] "p8a: Checking sleep_mammals is a dataframe."
```

```
## Test ##
test_that("p8b", {
  expect_true(ncol(sleep_mammals) == 11)
  print("p8b: Checking sleep_mammals has 11 columns.")
})
```

```
## [1] "p8b: Checking sleep_mammals has 11 columns."
```

```
## Test ##
test_that("p8c", {
  expect_true(nrow(sleep_mammals) == 3)
  print("p8c: Checking sleep_mammals has 3 rows.")
})
```

```
## [1] "p8c: Checking sleep_mammals has 3 rows."
```

9. [1 point] Suppose you are specifically interested in the sleep of horses and giraffes. From the original `sleep` dataset, assign `sleep_hg` to a data frame for horses and giraffes only.

```r
sleep_hg <- sleep %>% filter(name %in% c("Horse", "Giraffe")) #SOLUTION
```

```
BEGIN QUESTION
name: p9
manual: false
points: 1
```

```r
## Test ##
test_that("p9a", {
  expect_true(is.data.frame(sleep_hg))
  print("p9a: Checking sleep_hg is a dataframe.")
})
```

```
## [1] "p9a: Checking sleep_hg is a dataframe."
```

```r
## Test ##
test_that("p9b", {
  expect_true(ncol(sleep_hg) == 11)
  print("p9b: Checking sleep_hg has 11 columns.")
})
```

```
## [1] "p9b: Checking sleep_hg has 11 columns."
```

```r
## Test ##
test_that("p9c", {
  expect_true(nrow(sleep_hg) == 2)
  print("p9c: Checking sleep_hg has 2 rows.")
})
```

```
## [1] "p9c: Checking sleep_hg has 2 rows."
```

```r
## Test ##
test_that("p9d", {
  expect_true("Horse" %in% sleep_hg$name &&
                    "Giraffe" %in% sleep_hg$name)
  print("p9d: Checking sleep_hg has the correct rows.")
})
```

```
## [1] "p9d: Checking sleep_hg has the correct rows."
```

10. [1 point] From the original dataset,order the dataset by sleep time from shortest sleep time to longest sleep time. Assign this to `sleep_time`.

```
sleep_time <- sleep %>% arrange(sleep_total) #SOLUTION
```

```
BEGIN QUESTION
name: p10
manual: false
points: 1
```

```
## Test ##
test_that("p10a", {
  expect_true(is.data.frame(sleep_time))
  print("p10a: Checking sleep_time is a dataframe.")
})
```

```
## [1] "p10a: Checking sleep_time is a dataframe."
```

```
## Test ##
test_that("p10b", {
  expect_true(ncol(sleep_time) == 11)
  print("p10b: Checking sleep_time has 11 columns.")
})
```

```
## [1] "p10b: Checking sleep_time has 11 columns."
```

```
## Test ##
test_that("p10c", {
  expect_true(nrow(sleep_time) == 83)
  print("p10c: Checking sleep_time has 83 rows.")
})
```

```
## [1] "p10c: Checking sleep_time has 83 rows."
```

11. [1 point] Now order for longest sleep time to shortest sleep time. Assign this to `sleep_rev`.

```
sleep_rev <- sleep %>% arrange(sleep_total) #SOLUTION
```

```
BEGIN QUESTION
name: p11
manual: false
points: 1
```

```
## Test ##
test_that("p11a", {
  expect_true(is.data.frame(sleep_rev))
  print("p11a: Checking sleep_rev is a dataframe.")
})
```

```
## [1] "p11a: Checking sleep_rev is a dataframe."
```

```
## Test ##
test_that("p11b", {
  expect_true(ncol(sleep_rev) == 11)
  print("p11b: Checking sleep_rev has 11 columns.")
})
```

```
## [1] "p11b: Checking sleep_rev has 11 columns."
```

```
## Test ##
test_that("p11c", {
  expect_true(nrow(sleep_rev) == 83)
  print("p11c: Checking sleep_rev has 83 rows.")
})
```

```
## [1] "p11c: Checking sleep_rev has 83 rows."
```

12. [**2 points**] **Suppose you are interested in the order of sleep time, but according to whether the animal is a carnivore, herbivore, or omnivore. Rewrite the above statement to order sleep time according to the type of "-vore" that then animal is. Call this "sleep_time_rev":**

```r
sleep_time_rev <-  sleep %>% arrange(vore, -sleep_total) #SOLUTION
```

```
BEGIN QUESTION
name: p12
manual: false
points: 2
```

```r
## Test ##
test_that("p12a", {
  expect_true(is.data.frame(sleep_time_rev))
  print("p12a: Checking sleep_time_rev is a dataframe.")
})
```

```
## [1] "p12a: Checking sleep_time_rev is a dataframe."
```

```r
## Test ##
test_that("p12b", {
  expect_true(ncol(sleep_time_rev) == 11)
  print("p12b: Checking sleep_time_rev has 11 columns.")
})
```

```
## [1] "p12b: Checking sleep_time_rev has 11 columns."
```

```r
## Test ##
test_that("p12c", {
  expect_true(nrow(sleep_time_rev) == 83)
  print("p12c: Checking sleep_time_rev has 83 rows.")
})
```

```
## [1] "p12c: Checking sleep_time_rev has 83 rows."
```

13. [1 point] Create a new column called `rem_proportion` which is the ratio of rem sleep to total amount of sleep. Assign this new data frame to `sleep_ratio` from `sleep` data.

```
sleep_ratio <- sleep %>% mutate(rem_proportion = sleep_rem/sleep_total) #SOLUTION
```

```
BEGIN QUESTION
name: p13
manual: false
points: 1
```

```
## Test ##
test_that("p13a", {
  expect_true(is.data.frame(sleep_ratio))
  print("p13a: Checking sleep_ratio is a dataframe.")
})
```

```
## [1] "p13a: Checking sleep_ratio is a dataframe."
```

```
## Test ##
test_that("p13b", {
  expect_true(ncol(sleep_ratio) == 12)
  print("p13b: Checking sleep_time_rev has 12 columns.")
})
```

```
## [1] "p13b: Checking sleep_time_rev has 12 columns."
```

```
## Test ##
test_that("p13c", {
  expect_true(nrow(sleep_ratio) == 83)
  print("p13c: Checking sleep_ratio has 83 rows.")
})
```

```
## [1] "p13c: Checking sleep_ratio has 83 rows."
```

14. [1 point] Add a second column called `bodywt_grams` which is the bodywt column in grams.

```
sleep_r_bw <- sleep %>% mutate(rem_proportion = sleep_rem/sleep_total, bodywt_grams = bodywt * 1000) #S
```

```
BEGIN QUESTION
name: p14
manual: false
points: 1
```

```
## Test ##
test_that("p14a", {
  expect_true(is.data.frame(sleep_r_bw))
  print("p14a: Checking sleep_r_bw is a dataframe.")
})
```

```
## [1] "p14a: Checking sleep_r_bw is a dataframe."
```

```
## Test ##
test_that("p14b", {
  expect_true(ncol(sleep_r_bw) == 13)
  print("p14b: Checking sleep_r_bw has 13 columns.")
})
```

```
## [1] "p14b: Checking sleep_r_bw has 13 columns."
```

```
## Test ##
test_that("p14c", {
  expect_true(nrow(sleep_r_bw) == 83)
  print("p14c: Checking sleep_r_bw has 83 rows.")
})
```

```
## [1] "p14c: Checking sleep_r_bw has 83 rows."
```

15. [1 point] Calculate the average sleep time across all the animals in the dataset using a `dplyr` function and assign it to the variable `avg_sleep_time`. Your answer should be a data frame of 1 observation and 1 variable called `sleep_avg`

```r
avg_sleep_time <- sleep %>% summarize(sleep_avg = mean(sleep_total)) #SOLUTION
```

```
BEGIN QUESTION
name: p15
manual: false
points: 1
```

```r
## Test ##
test_that("p15a", {
  expect_true(is.data.frame(avg_sleep_time))
  print("p15a: Checking avg_sleep_time is a dataframe.")
})
```

```
## [1] "p15a: Checking avg_sleep_time is a dataframe."
```

```r
## Test ##
test_that("p15b", {
  expect_true(ncol(avg_sleep_time) == 1 &&
                  nrow(avg_sleep_time) == 1)
  print("p15b: Checking avg_sleep_time has 1 row and 1 column.")
})
```

```
## [1] "p15b: Checking avg_sleep_time has 1 row and 1 column."
```

```r
## Test ##
test_that("p15c", {
  expect_true(is.numeric(avg_sleep_time$sleep_avg))
  print("p15c: Checking sleep_avg column is a numeric.")
})
```

```
## [1] "p15c: Checking sleep_avg column is a numeric."
```

```r
## Test ##
test_that("p15d", {
  expect_true(all.equal(avg_sleep_time$sleep_avg, 10.43373, tol = 0.01))
  print("p15d: Checking sleep_avg column is 10.4337.")
})
```

```
## [1] "p15d: Checking sleep_avg column is 10.4337."
```

16. [2 points] Calculate the average sleep time for each type of "-vore". Hint: you'll need to use two `dplyr` functions! The column names should be `vore` and `sleep_avg`. Call this dataframe avg_by_vore

```
. = " # BEGIN PROMPT
avg_by_vore <- NULL # YOUR CODE HERE
" # END PROMPT

# BEGIN SOLUTION NO PROMPT
avg_by_vore <- sleep %>%
                 group_by(vore) %>%
                  summarize(sleep_avg = mean(sleep_total))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
# END SOLUTION
```

```
BEGIN QUESTION
name: p16
manual: false
points: 1
```

```
## Test ##
test_that("p16a", {
  expect_true(is.data.frame(avg_by_vore))
  print("p16a: Checking avg_by_vore is a dataframe.")
})
```

```
## [1] "p16a: Checking avg_by_vore is a dataframe."
```

```
## Test ##
test_that("p16b", {
  expect_true(ncol(avg_by_vore) == 2 &&
               nrow(avg_by_vore) == 5)
  print("p16b: Checking avg_by_vore has 5 rows and 2 columns.")
})
```

```
## [1] "p16b: Checking avg_by_vore has 5 rows and 2 columns."
```

```
## Test ##
test_that("p16c", {
  expect_true(identical(names(avg_by_vore), c("vore", "sleep_avg")))
  print("p16c: Checking column names are vore and sleep_avg.")
})
```

```
## [1] "p16c: Checking column names are vore and sleep_avg."
```

END