In-class worksheet 6

Feb 7, 2019

In this worksheet, we will continue to work with the tidyverse libraries:

```
library(tidyverse)
```

1. The msleep dataset

The `msleep` dataset, provided with ggplot2, contains information about sleep and awake times of different mammals:

```
msleep
```

```
## # A tibble: 83 x 11
##    name  genus vore  order conservation sleep_total sleep_rem sleep_cycle
##    <chr> <chr> <chr> <chr> <chr>              <dbl>     <dbl>       <dbl>
##  1 Chee… Acin… carni Carn… lc                  12.1      NA        NA    
##  2 Owl … Aotus omni  Prim… <NA>                17         1.8      NA    
##  3 Moun… Aplo… herbi Rode… nt                  14.4       2.4      NA    
##  4 Grea… Blar… omni  Sori… lc                  14.9       2.3       0.133
##  5 Cow   Bos   herbi Arti… domesticated         4         0.7       0.667
##  6 Thre… Brad… herbi Pilo… <NA>                14.4       2.2       0.767
##  7 Nort… Call… carni Carn… vu                   8.7       1.4       0.383
##  8 Vesp… Calo… <NA>  Rode… <NA>                 7        NA        NA    
##  9 Dog   Canis carni Carn… domesticated        10.1       2.9       0.333
## 10 Roe … Capr… herbi Arti… lc                   3        NA        NA    
## # … with 73 more rows, and 3 more variables: awake <dbl>, brainwt <dbl>,
## #   bodywt <dbl>
```

Verify that the sum of total sleep time (column `sleep_total`) and total awake time (column `awake`) adds up to 24h for all animals in the `msleep` dataset.

```
(msleep$sleep_total + msleep$awake) == 24
```

```
##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [12]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [23]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
## [34]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [45]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [56]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [67]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [78]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

There are two cases where the sum is not equal to exactly 24 hours.

```
msleep %>%
  mutate(day_total = sleep_total + awake) %>%
  filter(day_total != 24) %>%
  select(name, vore, sleep_total, awake, day_total)
```

```
## # A tibble: 2 x 5
##   name            vore  sleep_total awake day_total
##   <chr>           <chr>       <dbl> <dbl>     <dbl>
## 1 Pilot whale     carni         2.7  21.4      24.0
## 2 Common porpoise carni         5.6  18.4      24.0
```

In the pilot whale and the common porpoise, the total sleep and awake times add up to 24.05 hours.

Make a list of all the domesticated species in the `msleep` dataset, in alphabetical order. Hint: Domesticated species have the entry “domesticated” in the column `conservation` .

```
msleep %>%
  filter(conservation == "domesticated") %>%
  select(name) %>%
  arrange(name)
```

```
## # A tibble: 10 x 1
##    name
##    <chr>
##  1 Chinchilla
##  2 Cow
##  3 Dog
##  4 Domestic cat
##  5 Donkey
##  6 Guinea pig
##  7 Horse
##  8 Pig
##  9 Rabbit
## 10 Sheep
```

For the different vore classifications, tally how many species are awake for at least 18 hours. Hint: Use the function `tally()` .

```
msleep %>%
  filter(awake >= 18) %>%
  group_by(vore) %>%
  tally()
```

```
## # A tibble: 3 x 2
##   vore      n
##   <chr> <int>
## 1 <NA>      1
## 2 carni     4
## 3 herbi    11
```

Using the function `top_n()` , identify the top-10 least-awake animals and list them from least awake to most awake. Explain why this analysis gives you 11 results instead of 10. Hint: Before calling `top_n()` , use the function `select()` to extract the two columns `name` and `sleep_total` , in that order.

```
msleep %>%
  select(name, sleep_total) %>%
  top_n(10) %>%
  arrange(desc(sleep_total))
```

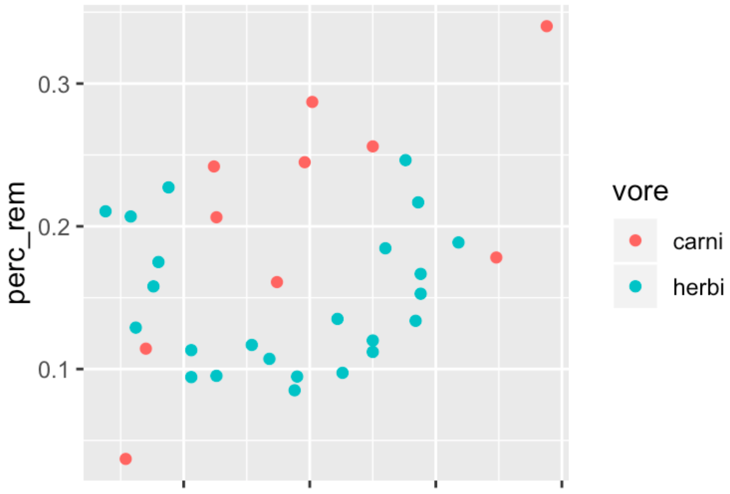
```
## Selecting by sleep_total
## # A tibble: 11 x 2
##    name                           sleep_total
##    <chr>                                <dbl>
##  1 Little brown bat                      19.9
##  2 Big brown bat                         19.7
##  3 Thick-tailed opposum                  19.4
##  4 Giant armadillo                       18.1
##  5 North American Opossum                18
##  6 Long-nosed armadillo                  17.4
##  7 Owl monkey                            17
##  8 Arctic ground squirrel                16.6
##  9 Golden-mantled ground squirrel        15.9
## 10 Tiger                                 15.8
## 11 Eastern american chipmunk             15.8
```

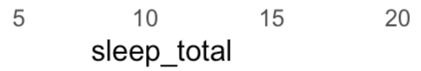
There are 11 results because there is a tie. Both the Tiger and the Eastern american chipmunk have a total sleep time of 15.8h, and they are in positions 10 and 11 of the list. Note that by default, `top_n()` orders based on the last variable in the table. Since we selected `sleep_total` as the last column before we called `top_n()` , we get the desired result.

Considering only carnivores and herbivores, make a plot of the percent of time each animal is in REM sleep (out of the total sleep time) vs. the animal's total sleep time. Hint: Use the operator `|` to indicate logical OR in the `filter()` function.

```
msleep %>%
  filter(vore == "carni" | vore == "herbi") %>%
  mutate(perc_rem = sleep_rem / sleep_total) %>%
  ggplot(aes(x = sleep_total, y = perc_rem, color = vore)) +
  geom_point()
```

```
## Warning: Removed 17 rows containing missing values (geom_point).
```





2. The diamonds dataset

The `diamonds` dataset provided by ggplot2 provides information about quality and price of 53940 diamonds:

```
head(diamonds)
```

```
## # A tibble: 6 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1 0.23  Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
## 2 0.21  Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
## 3 0.23  Good      E     VS1      56.9    65   327  4.05  4.07  2.31
## 4 0.290 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
## 5 0.31  Good      J     SI2      63.3    58   335  4.34  4.35  2.75
## 6 0.24  Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
```

The best cuts of diamonds are “Very Good”, “Premium”, and “Ideal”. Make a table that selects only those diamonds, and find the minimum, median, and maximum price for each cut. Hint: The operator `%in%` is helpful for selecting the diamond cuts.

```
diamonds %>%
  filter(cut %in% c("Very Good", "Premium", "Ideal")) %>%
  group_by(cut) %>%
  summarize(
    min_price = min(price),
    med_price = median(price),
    max_price = max(price)
  )
```

```
## # A tibble: 5 x 4
##   cut       min_price med_price max_price
##   <ord>         <dbl>     <dbl>     <dbl>
## 1 Fair            Inf        NA      -Inf
## 2 Good            Inf        NA      -Inf
## 3 Very Good       336      2648     18818
## 4 Premium         326      3185     18823
## 5 Ideal           326      1810     18806
```

For each of the different diamond cuts, calculate the mean carat level among the diamonds whose price falls within 10% of the most expensive diamond for that cut.

```
diamonds %>%
  group_by(cut) %>%
  filter(price > 0.9 * max(price)) %>%
  summarize(mean_carat = mean(carat))
```

```
## # A tibble: 5 x 2
##   cut       mean_carat
##   <ord>          <dbl>
## 1 Fair            2.73
## 2 Good            2.08
## 3 Very Good       1.97
## 4 Premium         2.08
## 5 Ideal           1.99
```

For each of the different diamond cuts, calculate the mean carat level among the top-10% most expensive diamonds.

```
diamonds %>%
  group_by(cut) %>%
  mutate(price_rank = rank(desc(price))) %>% # rank diamonds by price, in desce
nding order
  filter(price_rank < 0.1 * max(price_rank)) %>% # pick the top 10%
  summarize(mean_carat = mean(carat))
```

```
## # A tibble: 5 x 2
##   cut       mean_carat
##   <ord>          <dbl>
## 1 Fair            2.04
## 2 Good            1.76
## 3 Very Good       1.71
## 4 Premium         1.88
## 5 Ideal           1.58
```

Make a table that contains the median price for each combination of `cut` and `clarity`, and arrange the final table in descending order of median price.

```
diamonds %>%
  group_by(cut, clarity) %>%
  summarize(med_price = median(price)) %>%
  arrange(desc(med_price))
```

```
## # A tibble: 40 x 3
## # Groups:   cut [5]
##    cut       clarity med_price
##    <ord>     <ord>       <dbl>
##  1 Premium   SI2         4291
##  2 Ideal     SI2         4060.
##  3 Very Good SI2         4042
##  4 Good      SI2         3770
##  5 Fair      SI2         3681
##  6 Ideal     I1          3674.
##  7 Premium   SI1         3618
##  8 Fair      SI1         3528.
##  9 Very Good I1          3283
## 10 Premium   I1          3261
## # … with 30 more rows
```

Now arrange the same table first by cut and then within each cut group by median price.

```
diamonds %>%
  group_by(cut, clarity) %>%
  summarize(med_price = median(price)) %>%
  arrange(desc(cut), desc(med_price))
```

```
## # A tibble: 40 x 3
## # Groups:   cut [5]
##    cut     clarity med_price
##    <ord>   <ord>       <dbl>
##  1 Ideal   SI2         4060.
##  2 Ideal   I1          3674.
##  3 Ideal   SI1         2537
##  4 Ideal   VS1         1813
##  5 Ideal   VS2         1689
##  6 Ideal   VVS2        1330
##  7 Ideal   VVS1        1114
##  8 Ideal   IF          1022.
##  9 Premium SI2         4291
## 10 Premium SI1         3618
## # … with 30 more rows
```

3. If this was easy

For the `diamonds` data set, separately for each diamond cut, calculate the percentage of diamonds with a price above $10,000, and the median carat value for diamonds priced $10,000 or more.

```
diamonds %>%
  group_by(cut) %>%
  summarize(
    percent_above_10k = round(100 * sum(price >= 10000) / n(), 1),
    median_carat_above_10k = median(carat[price >= 10000])
  )
```

```
## # A tibble: 5 x 3
##   cut       percent_above_10k median_carat_above_10k
##   <ord>                 <dbl>                  <dbl>
## 1 Fair                    9.1                   2.01
## 2 Good                    7.6                   2
## 3 Very Good               9.3                   1.7
## 4 Premium                13.1                   1.72
## 5 Ideal                   8.2                   1.57
```