exercise_3

FOR/STT 875, Exercise 3

Learning objectives

- Create, subset, and manipulate vector contents and attributes
- Summarize vector data using R `table` and other functions
- Generate basic graphics using vector data

Overview

Modify this R Markdown file by filling in the code needed to answer the questions. The Exercise 3 video provides some guidance for completing this exercise.

Submission instructions

Upload your exercise_3.Rmd and exercise_3.html files to the Exercise 3 D2L dropbox.

Grading

You will receive full credit if your R Markdown document: 1) compiles without error; and 2) correctly completes the `# TODO:` code chunks. Also, please, fill in the feedback questions at the end of the exercise.

Getting started

First we'll set code chunk options to be applied to the entire document (i.e., global options). These global options can be overridden by the options in an individual code chunk. Typically we would use `echo = FALSE` in this chunk, since we wouldn't want it seen in the final document, but here we display the chunk. The global knitr options change how the R code is displayed in the resulting html document. Specifically, `comment = NA` removes the `##` comment that come before printed code and `tidy = TRUE` causes the printed code to wrap neatly.

```
knitr::opts_chunk$set(comment = NA, tidy = TRUE)
```

Zürichberg Forest inventory data set

The data come from the Zürichberg Forest which overlooks Lake Zürich in Switzerland. These data comprise

The data come from the Zürichberg Forest which overlooks Lake Zürich in Switzerland. These data comprise a complete enumeration of all trees in the Zürichberg Forest including species (`spp`), diameter at breast height (`dbh` ; the tree's diameter in cm at 1.4 m from the ground), and volume (`vol` ; the volume of the tree's stem in m^3).

We'll read in three vectors that hold tree species, diameter, and volume. The element ordering in the vectors correspond to the same tree (i.e., element 1 in each vector describes the first tree's characteristics).

Read in the Zürichberg Forest data.

```
con <- url("http://blue.for.msu.edu/FOR875/data/ZF_trees.gz")
load(con)
close(con)
rm(con)
ls()
```

```
[1] "dbh" "spp" "vol"
```

List the first 25 species in `spp` .

```
# TODO 3.1: List the species here
```

List the first 25 diameter measurements in `dbh` .

```
# TODO 3.2: List the DBH here
```

List the first 25 volume measurements in `vol` .

```
# TODO 3.3: List the volume here
```

The `table` function builds a contingency table. For example,

```
animals <- c("dog", "cat", "dog", "cat", "horse", "dog")
table(animals)
```

```
animals
  cat   dog horse
    2     3     1
```

Create a table that gives the number of times each species occurred in the data set.

```
# TODO 3.4: Show the table here
```

By default the table is ordered alphabetically by the name of the species. It might be more useful to order by the number of times a species occurs. Apply the `sort` function to the table to achieve this and assign the resulting table to a new variable called `spp_table` .

```
# TODO 3.5: Show the sorted table here
```

By observation, which three species are most abundant?

```
# TODO 3.6: Write the three most abundant spp here (comment out your answer
# using the # symbol)
```

Subset `spp_table` using `[]` to create a table called `top_spp` that contains the three most abundant species. Write your subset code so that it will always return the three most abundant species, even if the number of species changes. Do not hard code `6:8` in the square brackets, such as `spp_table[6:8]`; rather, use the `length` function and some basic math in place of hard coded vector positions.

```
# TODO 3.7: Create the top_spp table here
```

Draw a bar graph of the `top_spp` table using the `barplot` function.

```
# TODO 3.8: Create the bar graph here
```

Now add a y-axis label “Number of trees” to the barplot.

```
# TODO 3.9: Create the bar graph with y-axis labels here
```

Now make a table of species abundance, and sort in decreasing order, by giving the `decreasing = TRUE` argument in the `sort` function.

```
# TODO 3.10: Show the sorted table here
```

Use an R function to figure out how many trees are in the Zürichberg Forest data.

```
# TODO 3.11: Compute the answer here
```

The vectors listing tree diameter and volume would be more informative if the species were included. We can give names to each vector element using the `names` function. For example:

```
animals
```

```
[1] "dog"   "cat"   "dog"   "cat"   "horse" "dog"
```

```
names(animals) <- c("Asta", "Felix", "Snowy", "Garfield", "Mr. Ed", "Spud")
animals
```

```
    Asta    Felix    Snowy Garfield   Mr. Ed     Spud
   "dog"    "cat"    "dog"    "cat"  "horse"    "dog"
```

Use `names` to add the species as names to the `dbh` and `vol` vectors, and list the first ten elements of each vector with the names included.

```
# TODO 3.12: Create the vectors with names and list the first ten elements
# of each here.
```

More advanced subsetting

So far we have directly specified the elements of vectors that we want to extract, for example `spp[1]` or `dbh[length(dbh)]` or `spp[c(1,2,3,5)]`. More commonly we want to extract elements that meet a condition, such as all trees greater than some minimum DBH or all trees of a given species. For this we use subsetting with logical vectors, see Section 4.8 in the course book.

First some very simple examples using vectors of random numbers, and then you'll use these ideas for the tree vectors.

```
set.seed(123)  #Set the seed for reproducible random number generation
x <- rbinom(10, 20, 0.9)  #Generate 10 binomial(20, 0.9) values
y <- rbinom(10, 20, 0.8)  #Generate 10 binomial(20, 0.8) values
x
```

```
 [1] 19 17 18 16 16 20 18 16 18 18
```

```
y
```

```
 [1] 13 16 15 16 18 14 17 19 17 13
```

```
x[x > 16]  #Values of x which are larger than 16
```

```
[1] 19 17 18 20 18 18 18
```

```
x[x == 18]  #Values of x which are equal to 18
```

```
[1] 18 18 18 18
```

```
length(x[x == 18])  #How many values of x are equal to 18?
```

```
[1] 4
```

```
x[y < 15]  #Values of x for which the corresponding values of y are less than 15
```

```
[1] 19 20 18
```

```
x[x < y]  #Values of x which are less than the corresponding values of y
```

```
[1] 16 16
```

```
x[x == 18] <- 0  #Replace 18 by 0 in x
x
```

```
 [1] 19 17  0 16 16 20  0 16  0  0
```

Now we use logical subsetting to learn about the Zürichberg Forest data.

First, use logical subsetting to count the number of maple trees in the forest.

```
# TODO 3.13: Count the number of maple trees here
```

Did you answer the above question using `sum(spp=="maple")` ? It's okay if you didn't, there are several ways to count the trees that meet this condition. If you didn't use `sum(spp=="maple")` , then figure out why the `sum` function returns the correct answer.

A bit more on logical operators and subsetting

So far, we have made use of some comparison operators, including:

- Equal: `==`
- Not equal: `!=`
- Greater than: `>`
- Less than: `<`
- Greater than or equal to: `>=`
- Less than or equal to: `<=`

There are some logical operators we haven't seen yet, including the "and" operator and the "or" operator.

- and: `&`
- or: `|`

The `&` operator compares vector elements on its left and right to see if they match. If they are both `TRUE` , then `&` returns `TRUE` , otherwise `FALSE` . The `|` operator compares vector elements on its left and right to see if either of them are `TRUE` . If at least one is `TRUE` then `|` returns `TRUE` , otherwise if both are `FALSE` then `FALSE` is returned. These operations are done for each element pair along the vectors. For example:

```
c(FALSE, TRUE, FALSE) | c(TRUE, FALSE, FALSE)
```

```
[1]  TRUE  TRUE FALSE
```

```
c(FALSE, TRUE, FALSE) & c(TRUE, TRUE, FALSE)
```

```
[1] FALSE  TRUE FALSE
```

Say, you have a setting like that presented in TODO 3.16. You would like to extract elements from a given vector `a` if elements in either vector `b` or `e` are `TRUE` .

```
a <- 1:5

b <- c(TRUE, TRUE, TRUE, FALSE, FALSE)
e <- c(TRUE, FALSE, FALSE, FALSE, TRUE)

a[b | e]
```

```
[1] 1 2 3 5
```

Hint: in TODO 3.16 you need to do an additional step to create the logical vectors `b` and `e`, e.g., vector `b` could be `spp == "larch"`. There is more about logical operators in Section 8.2.1 in our book (although it should not be needed to complete this exercise).

Another useful logical operator is the `!` (i.e., the exclamation point, referred to as the "bang" in coding slang) which negates or flips the logical value, so for example `!=` represents not equal to.

Yet another very handy operator is `%in%` which is used to identify if an element occurs in a second vector. Consider the example below and consult the manual page via `help("%in%")`.

```
letters
```

```
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q"
[18] "r" "s" "t" "u" "v" "w" "x" "y" "z"
```

```
letters %in% c("a", "m", "q", "s")
```

```
 [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[12] FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE  TRUE FALSE FALSE FALSE FALSE
[23] FALSE FALSE FALSE FALSE
```

Use these ideas to:

Get the DBH of larch and silver fir trees and assign them to `dbh_l_sf` (you don't have to print out the resulting values, just assign to `dbh_l_sf`). Use the `%in%` operator for subsetting.

```
# TODO 3.14: Answer to question here
```

Using your solution from TODO 3.14, calculate the mean `mean` and standard deviation `sd` of the larch and sliver fir `dbh_l_sf` vector.

```
# TODO 3.15: Answer to question here
```

Repeat part TODO 3.14, but this time use the `|` operator (i.e., you should be able to get the same answer as TODO 3.14).

```
# TODO 3.16: Answer to question here
```

What is the maximum DBH and volume among the beech trees?

```
# TODO 3.17: Answer to question here
```

Congratulations! You've reached the end of Exercise 3.

Questions?

Reflect in ePortfolio | Download | Print

Activity Details

You have viewed this topic

Last Visited Jul 17, 2019 8:40 PM