STAT 19000 Project 4

Topics: user defined functions

Motivation: Despite the innumerous functions and packages R developers and users have created, often we need to create our own functions that solve a very specific problem. We demonstrated how user defined functions can be useful in the previous couple of projects. Being able to define our own functions is a powerful tool to have. It enables us to automate tasks, develop a cleaner code base, and debug our code with greater ease.

Context: We've been working with functions in R to gain insights from data using the suite of apply functions. We are taking a deeper dive into user defined functions and some complementary scoping rules.

Scope: Scoping rules and user-defined functions in R.

You can find useful examples that walk you through relevant material here or on scholar: `/class/datamine/data/spring2020/s` It is highly recommended to read through these to help solve problems.

Use the template found here or on scholar: `/class/datamine/data/spring2020/stat19000project04template.ipynb` to submit your solutions.

After each problem, we've provided you with a list of keywords. These keywords could be package names, functions, or important terms that will point you in the right direction when trying to solve the problem, and give you accurate terminology that you can further look into online. You are not required to utilize all the given keywords. You will receive full points as long as your code gives the correct result.

Don't forget the very useful documentation shortcut `?`. To use, simply type `?` in the console, followed by the name of the function you are interested in.

You can also look for package documentation by using `help(package=PACKAGENAME)`.

Sometimes it can be helpful to see the source code of a defined function. To do so, type the function's name without the `()`.

Question 1: categorizing movies

We will continue to look at the rotten tomatoes data from the last project. Read in the file found on scholar here:

`/class/datamine/data/spring2020/rotten_tomatoes_reviews.csv`

into a dataframe called `reviews`. As well as the file found on scholar here:

`/class/datamine/data/spring2020/rotten_tomatoes_movies.csv`

into a dataframe called `movies`.

1a. *(1 pt)* Create a function that takes a score between 0 and 100 and categorizes it as 'Fresh' if the score is greater or equal to 60 and 'Rotten' if it is less than 60. Call this function `categorize_score`. Test your function using the `tomatometer_rating` for the movie `Shadow Company` found in the `movies` dataset. Your function should return `Fresh`.

Keywords: *which, function, if, ifelse*

1b. *(1 pt)* Create a function that takes two arguments, *score1* and *score2*, and returns their mean. Call this function `summarize_scores`. Test your function using the scores from `tomatometer_rating` and `audience_rating` for the movie `Flashback` found in the `movies` dataset. Your function should return `49.5`.

Keywords: *which, function, mean, c*

Note: *Make sure to remove NA when calculating the mean value.*

Hint: *Make sure to wrap score1 and score2 into a vector prior to using the* `mean` *function.*

1c. *(2 pts)* Make a new function called `choose_scores`. `choose_scores` accepts two arguments: `score_vector` (which is a vector containing *score1* and *score2*), and `categorize` (which is either `TRUE` or `FALSE`).

If `categorize` is `TRUE` then `choose_scores(score_vector, categorize)` should first use the function from (1b) to calculate the mean of the scores. Then, it should use the function from (1a) to return "Fresh" or "Rotten", based on that mean score you just calculated.

If `categorize` is `FALSE` then `choose_scores(score_vector, categorize)` should use the function from (1b) to return the mean of the scores. Make sure to wrap the scores into one vector prior to passing it to our `choose_scores` function.

This function can be useful in two different ways:

`choose_scores(30, TRUE)` should return "Rotten".

`choose_scores(c(30, 40))` should return "Rotten".

`choose_scores(c(30, 40), TRUE)` should return "Rotten".

`choose_scores(c(30, 40), FALSE)` should return 35.

`choose_scores(30, 40, FALSE)` should not work because the user forgot to wrap the 30 and 40 into a vector.

Test your revised function, `choose_scores`, to verify that the movie Flashback is "Rotten" (by default, or if you manually use `TRUE` for the second argument) and has average score 49.5 (if you use `FALSE` for the second argument).

Please make the `categorize` argument `TRUE` by default, in case the user forgets to specify the value of categorize.

Keywords: *which, function, mean*

1d. *(1 pt)* Use your newly created `choose_scores` from (1c) to calculate whether a movie is `Rotten` or `Fresh` for all movies in the `movies` dataset. Put this result into a table that shows the number of "Rotten" and "Fresh" movies. Use the `apply` function with the option `MARGIN=1`. When `MARGIN=1`, each row of our data is given as a vector of inputs to our function (which is what we want, i.e. the `tomatometer_rating` and `audience_rating` passed as a vector to our `choose_scores` function).

Keywords: *apply, table*

1e. *(2 pt)* For the movies in both datasets, compare our created categorical score with the `critic_icon` column from the `reviews` dataset. What percentage of the critics' `Fresh` reviews did our score classify as `Rotten`? What percentage of reviews did our classification match the `critic_icon` column?

The first step in solving this problem is to add our custom classifications to the movies dataframe, into a column called `our_scores`. You can do this by using `apply` like in (1d).

The next step is to combine our `movies` and `reviews` dataset using a function called `merge`:

```r
subset_combined_dat <- merge(reviews[,c('rotten_tomatoes_link', 'critic_icon')],
                             movies[,c('rotten_tomatoes_link', 'our_scores')],
                             by='rotten_tomatoes_link')
```

What this does is it takes the `rotten_tomatoes_link` column and the `critic_icon` column from the reviews data frame, and the `rotten_tomatoes_link` column and the `our_scores` column from the movies data frame, and it combines them into a single data frame where `our_scores` are appended to each row in the reviews data frame based on the `rotten_tomatoes_link`.

```
  rotten_tomatoes_link critic_icon our_scores
1           /m/0814255      Rotten     Rotten
2           /m/0814255       Fresh     Rotten
3           /m/0814255      Rotten     Rotten
4           /m/0814255      Rotten     Rotten
5           /m/0814255      Rotten     Rotten
6           /m/0814255       Fresh     Rotten
```

As you can see, for movie with id "/m/0814255" our_score is "Rotten". This "Rotten" score is duplicated for each occurence of the id "/m/0814255". From this point, you should be able to utilize the `table` function to answer our questions.

Keywords: *merge, table, prop.table*

Question 2: scoping functions

(3 pts)

Your co-workers suggested giving the critics' ratings a lower weight than the audience rating when combining the scores. They sent you the code below as a suggestion. However the code is not returning what they expected when they run it from top to bottom (every line). They've asked for your help debugging it. Fix the scoping issue in your co-workers R script. Please note that you will only need to modify a small part of two lines. Do not delete anything from the code. Here is a good resource to read about scoping.

For this problem please submit:

1. The fixed version of this code, in it's entirety.
2. A small explanation of what was wrong and why.

```r
# testing idea out
tomatometer_rating <- c(10, 90, 65)
audience_rating <- c(86, 55, 45)

# example of doing a weighted mean score
(tomatometer_rating*0.8+audience_rating*1.2)/2

# function to weight scores
weight_score <- function(score, type){
        if(type == 'critic') score <- score*0.8
        if(type == 'audience') score <- score*1.2
        return(score)
}

# for critics we will down-weight
type <- "critic"
weight_score(tomatometer_rating, type)

# for audience we will up-weight
type <- "audience"
weight_score(audience_rating, type)

my_score_conversion <- function(tomatometer_rating, audience_rating){
        # down-weight
        tomatometer_rating <- weight_score(tomatometer_rating, type)

        # up-weight
        audience_rating <- weight_score(audience_rating, type)

        return(rowMeans(cbind(tomatometer_rating, audience_rating), na.rm=TRUE))
}

new_scores <- my_score_conversion(movies$tomatometer_rating, movies$audience_rating)

# score for movie in row 123
new_scores[123] # should be 54.4
```

Keywords: *R scoping rules*

Hint: *Use the functions `formals`, `body`, and `enviroment` to get a better understanding of your co-workers functions.*

Project Submission:

Submit your solutions for the project at this URL: https://classroom.github.com/a/pFNrJXR- using the instructions found in the GitHub Classroom instructions folder on Blackboard.

Important note: Make sure you submit your solutions in both .ipynb and .pdf formats. We've updated our instructions to include multiple ways to convert your .ipynb file to a .pdf on scholar. You can find a copy of the instructions on scholar as well: `/class/datamine/data/spring2020/jupyter.pdf`. If for some reason the script does not work, just submit the .ipynb.