Feedback Assignment 508875

Summary

This week was all about enhancing your skills in R, especially with regard to using built-in functions, handling data types, managing directories, and applying statistical functions. Overall, you are progressing well, especially in understanding variables and built-in packages. However, there are areas where you can improve, such as sorting data before indexing and managing factors and NA values. Make sure to work on adding more detailed comments to clarify your code and avoid minor mistakes.

I recommend reviewing the feedback, particularly for questions R2, R11, R15, and R27 to refine your skills. To help address your current areas of improvement, I've created a personalized coding challenge for you.

Coding Challenge

We invite you to work on the following personalized coding challenge and submit your result on Canvas. Dont worry about being perfect, this is ungraded and just for your practice.

Coding Challenge:

1. Create a vector of randomly generated grades for 100 students.
2. Sort the grades in ascending order and select the 1st, 25th, 50th, 75th, and 100th percentiles.
3. Convert the grades to a factor labeled "Fail", "Pass".
4. Replace any NA values in the grades with the mean grade, recalculated without NA values.
5. Comment your code clearly to explain each step.

Hint: Use functions like `sample()`, `sort()`, `quantile()`, `factor()`, and `ifelse()`. As always, explore the R documentation and related resources to find solutions.

#R1

Great job using the `getwd()` function to determine your current working directory. You have successfully used a built-in function, aligning perfectly with the learning goals.

If you want to further explore this topic, consider asking the following question during the next lecture: *What are other useful functions for managing the working environment in R?*

--

#R2

Great job using the `setwd` function to set your working directory in R. However, to fully meet the task requirements, make sure to include the path up to the `week_1` subfolder. For example:

```
setwd("/Users/lukekorthals/Desktop/repos/pips-course/r_course/week_1")
```

This will ensure that the working directory is specifically set to the `week_1` subfolder, which was part of the task.

If you want to dig deeper, you might ask during the next lecture: *What are some common pitfalls when setting a working directory in R, and how can we troubleshoot them?*

--

#R3

Great work on utilizing the `list.files()` function and looking up its details using `? list.files`. You have demonstrated a good understanding of how to work with built-in functions in R and how to find more information about them.

If you want to learn more about managing files and directories in R, consider asking the following question during the next lecture: *What are some other useful functions for file management in R?*

--

#R4

Great work on understanding and applying the concept of variables and data types in R. Your code accurately captures the required assignments and demonstrates an understanding of why the addition fails.

Here's a small improvement for clarity:

1. **Use comments to elaborate further:** It might be useful to mention the `typeof()` function to check data types. This can be instructive for understanding why the addition fails:

```
    typeof(hours_sleep)
    typeof(reported_happiness)
```

If you are wondering more about data types in R and their operations, consider asking: *What are the different data types in R, and how do they behave in operations?*

--

#R5

Great job analyzing the code and correctly identifying the outcome. Your comments accurately describe what happens in each line.

Keep practicing with different functions and concepts to enhance your understanding of the R environment.

If you want to learn more about managing the workspace in R, you could ask: *What are some best practices for handling large numbers of objects in the R workspace?*

--

#R6

Excellent job! You successfully created the variable `classic_string`, returned its value, and used the `print()` function correctly. This shows a good understanding of variables and built-in functions in R.

If you have further questions, you might ask: *Can we use other functions to display variables in R apart from the `print()` function?* This would deepen your understanding of displaying variables.

--

#R7

Great job defining the vector and identifying the datatype transformation!

1. **Minor Improvement:** You can use `str(a)` in addition to `typeof(a)` to get more detailed information about the structure of the vector.

```
    str(a)
```

If you want to clarify this concept further during the lecture, consider asking: *Why do vectors in R coerce all elements to the same type, and how does this affect data processing?*

--

#R8

Well done on identifying the issue with the `NA` value and correctly using the `na.rm=TRUE`

parameter to compute the mean! Your code now successfully calculates the mean with the missing value removed:

```
participant_ages <- c(21,26,27,19,49,NA)
mean(participant_ages, na.rm=TRUE)
```

You've shown a good grasp of reading and understanding the function documentation.

If you'd like to dive deeper, consider asking in the next lecture: *Are there other statistical or mathematical functions where handling NA values is important?*

--

#R9

Great job installing and using the `fortunes` package in R! You correctly followed the steps to install the package, load it, and use the `fortune()` function with your birth-month. Everything works perfectly, and you've achieved all the learning goals of this task.

If you want to deepen your understanding of working with packages in R, consider asking the following question during the next lecture: *How can we update R packages and check for the latest available versions?*

--

#R10

Great job identifying the forbidden mathematical operation as division by zero. This understanding shows your comprehension of how logical values are treated in arithmetic in R.

If you have further questions, consider asking during the lecture: *What are the numerical values assigned to logical `TRUE` and `FALSE` in R?*

--

#R11

You're on the right track with generating and indexing the grades. However, you missed the step of sorting the grades before indexing.

Here's the corrected code:

```
grades = sort(rnorm(100, 7.5, 1))
grades[c(1, 20, 80)]
```

This ensures that your grades are sorted before you index them.

If you want further clarification on this concept, you could ask: *What are other efficient ways to sort and subset data in R?*

--

#R12

Your code accurately counts the number of letters in the alphabet using `length(LETTERS)`. Great job using built-in functions and data in R!

However, remember to explore the help documentation with `?LETTERS` as indicated in the task. This will help you understand how to access and utilize built-in datasets and their documentation in R, which is a valuable skill.

To further solidify your understanding, consider asking: *What other built-in constants or datasets are frequently used in R, and how can I access their documentation?*

--

#R13

You did a great job identifying that the second line is different because it performs a logical comparison instead of an assignment. This shows you have a good understanding of variables and logical operations in R. Keep up the good work!

If you want to deepen your understanding, you could ask: *How can logical operators be effectively used in R for data analysis?*

--

#R14

Good attempt at solving the problem using the `gsub` function in R. You are almost there! Here are a couple of pointers:

1. **Storing Results:** When you use `gsub`, ensure you store the result back into the `fruits` variable or a new variable to reflect the changes.

   ```
   fruits <- gsub("\\.", "a", fruits)
   ```

By making this change, the new `fruits` vector will have the corrected characters.

If you're curious about similar functions in R, you might ask: *What are the differences between gsub, sub, and str_replace functions for string manipulation in R?*

--

#R15

You are on the right track with your solution, especially by converting the factor and handling NA values. However, working with factors requires careful conversion. Here's how you can improve your code:

1. **Convert factor to numeric directly, including handling "NA" correctly:**

   ```
   horrible_numbers <- factor(c("25", pi, "NA", 1))
   ```

```
numeric_values <- as.numeric(as.character(horrible_numbers))
mean(numeric_values, na.rm=TRUE)
```

If you want to learn more about data types and conversions in R, you might ask: *What are the common pitfalls when converting factors to numeric in R?*

--

#R16

Your implementation is excellent and meets all the learning goals! You have correctly used variables, performed the angle conversion, and applied the Law of Cosines using built-in R functions.

If you would like further clarification on working with trigonometric functions in R, a useful question to ask during the lecture might be: *What are some common use-cases for trigonometric functions in real-world data analysis?*

--

#R17

Great job using built-in functions in R to generate a sequence and calculate the product efficiently. Your code is concise and correct, achieving the task's goal. Keep up the good work!

If you want to understand more about built-in functions in R, consider asking the following question during the next lecture: *What are some other useful built-in functions in R for mathematical operations?*

--

#R18

Great job setting the seed for reproducibility and using the `sample()` function to generate random dimensions for the matrix. Your code effectively creates a matrix and shows its dimensions. Here are a couple of tips to improve your code further:

1. **Explicitly fill the matrix with missing values (NA):** While your matrix is filled with `NA` by default, it's good practice to explicitly state this for clarity.

   ```
   m = matrix(NA, nrow = n_rows, ncol = n_cols)
   ```

This makes it clear to anyone reading your code that the matrix is intentionally filled with missing values.

If you want to dive deeper into handling complex data structures in R, consider asking the following question during the next lecture: *What are the benefits of using matrices versus data frames in R?*

--

#R19

Well done identifying that EEGData3 is incorrect because it has 201 rows and 1325 columns. Your understanding of how the order of arguments in functions matters is also accurate. Here are a few suggestions to improve your explanation:

1. **Specify function arguments explicitly:** To avoid mistakes, always specify arguments explicitly when dealing with functions that have multiple parameters.

   ```
   EEGData3 <- matrix(runif(n_subjects*n_vars), nrow=n_subjects, ncol=
   ```

2. **Check arguments when reading code:** When reading and debugging code, always check function arguments and their defaults if not specified.

Your comment about the order and necessity of explicitly specifying function arguments is a good learning point. Keep practicing to solidify your understanding of function arguments and matrix operations in R.

If you'd like to explore more, you could ask during the lecture: *What are some common pitfalls when creating matrices and data structures in R, and how can we avoid them?*

--

#R20

You did a good job in identifying the need to sum across the matrix, and correctly used vectorized functions. However, to get the number of correct responses per participant (row-wise summation), you should replace `colSums(M)` with `rowSums(M)` like this:

```
#WRITE YOUR CODE INTO THIS CELL (edit 1 line of code)
set.seed(1234)
rows <- 20 -> cols
M <- matrix(sample(c(T, F), rows*cols, replace = T), rows, cols)
rowSums(M)
```

This adjustment will compute the number of correct responses for each participant.

If you want to deepen your understanding of matrix operations in R, you could ask: *'What are other useful functions for operating on rows and columns of a matrix in R?'*

--

#R21

Great job creating a 2x2x2 array and labeling it using `dimnames`! You have successfully used complex data structures in R. Your code is clear and concise.

If you want to deepen your understanding of arrays and their applications, you could ask during the next lecture: *Can we perform arithmetic operations on arrays in R, and if so, how does R handle them?*

--

#R22

Excellent work on filling in the missing relationships in the matrix! You've shown a solid understanding of both position-based and name-based indexing in R. This is a key skill when working with matrices and data frames.

If you want to delve deeper into the topic, consider asking during the lecture: *"What are some other common methods of subsetting and indexing in R beyond using matrices?"*

--

#R23

You've made a good start by identifying the built-in function `diag` and understanding its role in matrix operations.

1. **Code Improvement:** Your final approach is accurate and works perfectly by directly assigning `NA` to the diagonal of the matrix:

   ```
   diag(relationship_matrix) = NA
   ```

2. **Understanding Indexing in R:** To make it clear, `diag(matrix)` accesses the diagonal elements directly as a vector, allowing simple assignment of missing values.

Good job on simplifying the task and making your code more readable. Next time, ensure to verify the output to confirm that your changes have the intended effect.

If you want to delve deeper into matrix manipulation, you might ask: *How can we access specific submatrices or rows and columns in an R matrix efficiently?*

--

#R24

Great job on solving the task using the `rep` function correctly. Your code is both precise and efficient, fulfilling all the task requirements and learning goals:

1. A correctly ordered vector with the desired sequence.
2. Effective use of the `rep` function.

There's no need for improvement in this case. Well done!

If you have questions about other potential uses of the `rep` function, consider asking during the lecture: *"What are some advanced use cases for the `rep` function in data manipulation?"*

--

#R25

Well done on utilizing `rep()` and `seq()` to create the dataframe. Here is a small tip to correct the pattern in the `Morality` column:

1. **Correct the pattern for the `Morality` column:** To match the desired output, you should repeat the correct sequence.

   ```
   Morality = rep(c("Bad", "Good", "Bad", "Good", "Bad", "Good"))
   ```

After making this change, the dataframe will align exactly with the given example.

If you have any further queries, you might ask during the next lecture: *How can I verify that my patterns in a dataframe are correct and match the expected output?*

--

#R26

Great job creating the final_grade column by correctly applying the weighted averages for assignment and exam grades! Your solution meets the task requirements and demonstrates a good understanding of basic data manipulation in R.

If you have any further questions related to this task, you might ask: "Can you explain more advanced ways to manipulate data frames in R for larger datasets?"

--

#R27

Your solution is almost perfect! You successfully created a boolean variable to indicate whether a student passed the class based on their final grade and exam grade. To fully complete the task, you need to add a step to print the IDs of the students who passed. Here's an updated version of your code:

```
grades$passed = grades$final_grade >= 5.5 & grades$exam >= 5.5
print(grades$id[grades$passed == TRUE])
```

This will create the `passed` column and print the IDs of the students who passed the class.

If you want to learn more about handling boolean operations, consider asking the following question during the next lecture: *How can I combine multiple logical conditions in R more efficiently? *

--

#R28

Excellent work! Your code achieves the desired task perfectly by exporting the InsectSprays dataset to a CSV file without row names.

If you want to dive deeper, consider asking the following question during the next lecture: *What other parameters can we use with the write.csv function in R?*

--

#R29

Great job working with packages and generating dynamic output in R! Here are a couple of tips to streamline your code:

1. **Avoid unnecessary package installations**: Package installation commands

(`install.packages()`) should usually be run separately outside of your script to avoid reinstalling packages each time you run the code. Your code after the packages are installed would look like this:

```
library(statquotes)
library(cowsay)

say(statquotes::statquote()$text, by = sample(names(cowsay::animals
```

2. **Improving readability**: Adding relevant comments can make your code easier to understand for others and yourself when you revisit it later.

If you want to dive deeper into working with R packages, consider asking the following question in your next lecture: *"Are there best practices for managing and installing R packages in different environments?"*

--

#R30

Excellent work! Your code correctly reads the dataset, subsets the data based on mood swings, and runs the t-test. This matches the task requirements perfectly and demonstrates your understanding of handling data structures and performing statistical analyses in R.

If you want to learn more about working with complex datasets, consider asking the following question during the next lecture: `What other types of data structures should I be familiar with when working with larger datasets in R?`

--

#Radv1

Good attempt at creating vectors in R. Here are some suggestions for improvement:

1. **Correct the `seq` function usage:** Instead of `seq(1:3)`, you should use `seq(1, 3)`:

   ```
   vec3 <- seq(1, 3)
   ```

2. **Refactor `rep` usage:** To be more concise:

   ```
   vec4 <- rep(c(1, 2, 3), times = 1)
   ```

3. **Preserve all vectors:** Ensure you create unique variable names for each vector to avoid overwriting:

   ```
   vec1 <- c(1, 2, 3)
   vec2 <- 1:3
   vec3 <- seq(1, 3)
   vec4 <- rep(c(1, 2, 3), times = 1)
   ```

4. **Check the data types of all vectors:**

   ```
   typeof(vec1)
   typeof(vec2)
   ```

```
    typeof(vec3)
    typeof(vec4)
```

Here's how your revised code might look:

```
vec1 <- c(1, 2, 3)
vec2 <- 1:3
vec3 <- seq(1, 3)
vec4 <- rep(c(1, 2, 3), times = 1)

typeof(vec1)
typeof(vec2)
typeof(vec3)
typeof(vec4)
```

If you want to learn more about vectors in R, consider asking the following question during the next lecture: *"What are the differences between `c()`, `:` and other vector creation functions in R?"*

--

#Radv2

Great job demonstrating the concept of indexing using the `cars` dataset. Your code accurately shows how to select a row, a variable, and a specific cell. Here are a couple of suggestions to improve your submission:

1. **Define the dataset being used:** It is a good practice to explicitly mention that you are using a built-in dataset and, optionally, provide a brief description.

   ```
   # Using the built-in 'cars' dataset in R
   data(cars)
   ```

2. **Add comments to improve readability:** Adding comments helps others (and yourself) understand the purpose of each line.

   ```
   # Select the first row
   cars[1, ]

   # Select the 'speed' variable (column)
   cars$speed

   # Select the first cell in the 'speed' column
   cars$speed[1]
   ```

Including these comments will make your code more readable and provide better context.

If you want to delve deeper into indexing and subsetting datasets, consider asking the following question during the next lecture: *"What are some advanced techniques for data subsetting in R?"*

--

#Radv3

You've done an excellent job installing the 'stringr' package and using `stringr::str_match_all` without calling `library()`. This demonstrates a good understanding of how to directly access functions from a package namespace.

If you want to further improve, consider adding a comment explaining your code. Clear comments can help others (and yourself) understand the purpose of different parts of your code quickly. Here's how to do it:

```
# Install the 'stringr' package
install.packages("stringr")

# Use the str_match_all function from the 'stringr' package to find all
stringr::str_match_all("alalalalala", "a")
```

If you have any questions about this task, feel free to ask: *Can you use the functions of another package in a similar way?* This might help you better understand the concept of using namespace in R.

--

#Radv4

Great job simulating the categorical and continuous variables! Your choice of normal distribution for the continuous variable and setting the seed for reproducibility are both excellent practices.

Here's how you can improve your code further:

1. **Create a dataframe:** You need to combine the variables into a dataframe to fulfill the task requirements.

   ```
   data <- data.frame(animal=animal, cuteness=cuteness)
   ```

2. **Combine your code:** Including the dataframe creation, your entire code will look like this:

   ```
   set.seed(1234)
   animal <- rep(c("cat", "dog"), each=100)
   cuteness <- c(rnorm(100, 4, 1), rnorm(100, 10, 2))
   data <- data.frame(animal=animal, cuteness=cuteness)
   ```

If you want to learn more about working with dataframes in R, consider asking the following question during the next lecture: *"What are some common operations for manipulating dataframes in R?"*

--

#Radv5

Good job creating and dealing with missing values in your dataset! Here are a few suggestions to improve your code:

1. **Add more comments**: Enhance your explanation for each step to match the code's logic, which will make it easier to understand.

```
# Creating a vector of random values with two NAs
values = c(rnorm(100), NA, NA)

# Creating a data frame with two columns: group and values
dat = data.frame(group = rep(c("A", "B"), each=51), values = values

# Imputing NAs in 'values' column with the mean of non-missing valu
dat$values[is.na(dat$values)] = mean(dat$values, na.rm=TRUE)
```

2. **Target specific columns for NA imputation**: Ensure you are only imputing NAs in specific columns.

```
# Imputing NAs only in the 'values' column
dat$values[is.na(dat$values)] = mean(dat$values, na.rm=TRUE)
```

If you want to learn more about different imputation methods, consider asking: *What are other imputation methods besides the mean, and when should we use them?*

--

#Radv6

You did an excellent job creating a named list and explaining its usefulness compared to a vector. Your variable names are descriptive and your comment accurately reflects the advantage of using a list in this context. This matches the learning goals of working with data structures, commenting, and understanding data types in R.

If you want to deepen your understanding, consider asking this question in the next lecture: *"What are some real-world scenarios where lists are preferred over vectors in R?"*

--

#Radv7

Great job finding an interesting discussion and clearly stating why you found it intriguing!

A couple of minor improvements:

1. **Avoid informal language**: "mind boggling" and "weird reasons" might be too casual. A more professional tone is beneficial for documentation and comments.

   ```
   # https://stackoverflow.com/questions/1741820/what-are-the-differen
   # The discussion on the differences between the "<-" and "=" assign
   ```

2. **Expand slightly on your reasoning**: You could include a bit more detail to explain why the discussion grabs your interest, such as potential performance implications or common misunderstandings, to show deep engagement with the topic.

   ```
   # https://stackoverflow.com/questions/1741820/what-are-the-differen
   # The discussion on the distinctions between the "<-" and "=" assig
   ```

These adjustments would add clarity and professionalism to your comment while still expressing your interest.

If you want to learn more about the rationale behind different coding conventions, consider asking the following question during the next lecture: *What are the benefits and drawbacks of different assignment operators in R and other programming languages?*

--

#Radv8

You're on the right track with reading a dataset from a URL and visualizing a specific column in R. Here are a few ways to improve your code:

1. **Add Axis Labels and Title:** Providing axis labels and a title can make your plot easier to understand.

   ```
   plot(dat$Sadness, main="Sadness Levels", xlab="Index", ylab="Sadnes
   ```

2. **Check for Missing Data:** Before plotting, it's good practice to check for and handle any missing data.

   ```
   sum(is.na(dat$Sadness)) # This will help you find out if there are
   ```

If you want to explore more about data visualization in R, you might consider asking: `What are some advanced plotting functions or libraries in R that offer more customization?`

--

#Python1

Great job identifying the `as.numeric()` function in R as the equivalent to `int()` in Python. Here's how you can write the equivalent code in R for better clarity:

```
# The Python equivalent code written in R
first_number <- 5
second_number <- "6"
first_number + as.numeric(second_number)
```

This ensures you demonstrate the full understanding of both languages by actually providing the translated code.

If you want to learn more, consider asking in the next lecture: *What are some other common type conversion functions in both R and Python?*

--

#Python2

You've correctly identified the use of pip to install Python packages, which aligns with the task. However, there's no need to activate a conda environment here. You can simplify your solution by directly calling pip install for numpy and pandas. Here's the improved version of the bash commands:

```
pip install numpy pandas
```

This command installs both packages in one line, making your code more concise.

Consider asking this during the next lecture: *When might we need to use a virtual environment in Python, and what are its benefits?*

--

#Python3

You did a perfect job resolving the `NameError` by correctly adding the line `import numpy as np`. This shows a good understanding of module imports and debugging error messages in Python. Well done!

If you want to delve deeper, consider asking during the lecture: *What are some common numpy functions and their uses in scientific computing?*

--

#Python4

Great job identifying and fixing the issue by using the `.copy()` method. Your explanation about pointers and creating a new copy is spot on. Keep practicing to strengthen your understanding of how variables and references work in Python, especially with mutable objects like numpy arrays.

If you have any further questions, you might ask: *What are the differences between shallow and deep copies in Python and when should we use them?*

--

#Python5

Great job identifying the issue with `np.mean` and using `np.nanmean` to compute the mean with the missing value removed. Your code is now correctly computing the mean while ignoring NaN values, and it is well-written. You have demonstrated a good understanding of handling missing values in numpy.

If you have further questions about handling missing values or any other numpy function during the lecture, consider asking: *What other numpy functions are available that handle NaN values?*

--

#Python6

Your code is perfect for creating a 3-dimensional array of 4 by 3 by 5 with all elements being zero using numpy. Well done!

If you need more insight into working with complex data structures using numpy, consider asking: *What other functions in numpy can be used to initialize different types of arrays?*

--

#Python7

Excellent job on using numpy to generate the series of odd numbers and compute their product. Your code is concise and correctly applies the `np.prod` and `np.arange` functions.

A small suggestion for further improvement:

  • Explicitly import numpy at the beginning of your script to avoid potential errors in a larger project or collaborative environment.

Example:

```
import numpy as np
print(np.prod(np.arange(1, 100, step=2)))
```

This will make your code more robust and easier to understand for others who might read it.

Would you like to deepen your understanding of how to work with numpy arrays and their different functions in Python?

--

#Python8

Your use of pandas to create the dataframe for LOTR characters is nearly perfect! You correctly used list multiplication for the 'First_appearance' column and `range` for the 'Dating_appeal' column. However, there is a small issue in how you generated values for the 'Morality' column. The current implementation creates an alternating pattern of ["Bad", "Good", "Bad", "Good", "Bad", "Good"], which doesn't match the provided table. Here's a slightly adjusted solution:

```
import pandas as pd

lotr_characters = pd.DataFrame({
    'Name': ["Gollum", "Smeagol", "Sauron", "Frodo", "Shelob", "Samwise
    'First_appearance': ["The Hobbit"]*3 + ["LOTR"]*3,
    'Morality': ["Bad", "Good", "Bad", "Good", "Bad", "Good"],
    'Dating_appeal': list(range(0, 11, 2))
})
```

By explicitly defining the 'Morality' column values, it adheres to the values given in the table. Otherwise, excellent work!

If you want to dive deeper, consider asking this question during the next lecture: *"What are other effective ways to create pandas dataframes when handling larger datasets?"*

--

#Python9

Your code correctly lists all filenames in your current working directory using

`os.listdir()`. However, it does not combine them into a single string as the task requires. You can use the `join()` function to achieve this. Here is an improved version of your code:

```
import os
print(" ".join(os.listdir()))
```

If you want to learn more about working with directories and handling strings, consider asking the following question during the next lecture: *What are some other useful functions in the os module for file and directory operations in Python?*

--

#Python10

Your solution meets all the requirements of the task and demonstrates a good understanding of numpy and Python data structures. Excellent job!

If you want to further enhance your code, you could add a seed for reproducibility, which is especially important in scientific experiments:

```
import numpy as np
np.random.seed(42)   # Setting the seed for reproducibility
my_measurements = {
  "my_mood_measurements": np.random.choice(["happy", "sad"], 365),
  "my_iq_measurements": np.random.normal(100, 15, 52)
}
my_measurements
```

This will ensure that every time you run the code, you get the same results.

If you want to learn more about simulating data in Python, consider asking the following question during the next lecture: *Why is it important to set a random seed when simulating data?*

--