DSI Data Science: Introduction to R Assessment 1

Introduction and Goals

Please carefully review all instructions provided. Throughout the course, we had the opportunity to familiarize ourselves with the RStudio environment, gain knowledge about its structure, utilize R syntax, discover methods for obtaining assistance, and make use of pre-existing functions. Additionally, we acquired the necessary skills to install R packages from CRAN, Bioconductor, and GitHub. Our exploration encompassed R data types and structures, followed by the installation of the tidyverse package (Wickham, H. et al, 2019) and an examination of its functions. The purpose of this assessment is to ensure that learners can accomplish the following:

- Recognize the structure of RStudio and navigate its environment proficiently.
- Comprehend the distinctions among various R data types and structures.
- Apply the rules of R coercion.
- Identify and handle missing values.
- Utilize built-in functions as well as functions from downloaded R packages.

The questions in the assessment carry a total of 30 marks, while an additional 3 marks are allocated for formatting (see Submission Instructions on page 5).

Total potential marks: /33

Tasks

Please respond to all the questions sequentially. You have the option to utilize a PDF editor, the provided R markdown template, or any other platform/software of your preference to generate the PDF document containing both the questions and answers. Alternatively, you can open your R script using a text editor and export it as a PDF document. When submitting your answers, make sure to include both the question number and the question itself, along with your responses. This requirement is in place to ensure that no questions are skipped and that all sub-questions within each main question are addressed. When copying and pasting the questions, you may disregard any formatting such as italics.

1. $1 mark$

   In RStudio, the (top/bottom , left/right) **TOP RIGHT** is where objects/variables and functions that are currently available in the R session are displayed.

2. $1 mark$

   In RStudio, the (top/bottom , left/right) **BOTTOM LEFT** is where users can input R commands, and it is where R displays the corresponding results of those commands.

3. $1 mark$

   In RStudio, the (top/bottom , left/right) **TOP LEFT** is where you can edit and run scripts.

4.

1*mark*

In RStudio, the (top/bottom , left/right) **BOTTOM RIGHT** is where you can view plots, files and help documentation.

1.

5*marks*

From the table below, state the type (character, logical, integer, double, raw, complex) for each element.

element	type
"j"	Character
"1"	Character
3i	Complex
1.7	Double
TRUE	Logical

5.

1*mark*

Create a vector named 'my_vector' containing numbers 1 to 10.

```
my_vector <- c(1:10)
#alternative
my_vector <- seq(1,10)
```

6.

1*mark*

Perform a mathematical operation that would double each number in 'my_vector' from the question above. Save this as a new variable named 'my_vector_doubled'

```
my_vector_doubled <- my_vector * 2
```

7.

1*mark*

Look up help for R base function mean() (Hint: type ?mean in the console). Exhibit the application of the mean function by utilizing it on the variable 'my_vector_doubled'.

```
mean(my_vector_doubled)
```

```
## [1] 11
```

8.

2*mark*

Coerce 'my_vector_doubled' into a character and store it as a new variable named 'character_vector'. What type of coercion occurs when creating the vector 'character_vector'; implicit or explicit?

```
character_vector <- as.character(my_vector_doubled)
```

This is an explicit coercion as it explicitly converts the numeric vector to a character vector using the `as.character()` function

9.

1*mark*

The "stringr" package is a popular package in R that provides a set of powerful functions for manipulating and working with strings. The package includes functions for tasks such as pattern matching, string extraction, replacement, splitting, and merging. The "stringr" package is a part of the "tidyverse" collection of R packages and widely used in data wrangling, text mining, and data analysis tasks in R. To use the "stringr" package along with other tidyverse packages, you can install and load the entire tidyverse package to your current R session. Alternatively, you can simply install and load the "stringr" package.

```
#install.packages("tidyverse")
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.2     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v ggplot2   3.4.2     v tibble    3.2.1
## v lubridate 1.9.2     v tidyr     1.3.0
## v purrr     1.0.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become e
```

10.

$1 marks$

The following vector named 'missing_values_vector' has been created. This vector was built using reserved term LETTERS, which contain all letters of the alphabet. Using the c() function, which is for concatenation, the letters have been combined with 3 missing values indicated by NA, as shown below. Write a code to detect missing values.

```
missing_values_vector <- c(LETTERS, NA,NA,NA)
missing_values_vector
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
## [20] "T" "U" "V" "W" "X" "Y" "Z" NA  NA  NA
```

```
is.na(missing_values_vector)
```

```
##  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [25] FALSE FALSE  TRUE  TRUE  TRUE
```

11.

$2 marks$

How would you identify which positions contain missing values in 'missing_values_vector' ? (Hint: There could be multiple ways to perform this. One method would be to use the which() and is.na() functions).

```
which(is.na(missing_values_vector))
```

```
## [1] 27 28 29
```

12.

$4 marks$

Fix the following errors:

```
#create a vector named my_numbers and autoprint it.
my_numbers <- c(1:5)
my_Numbers
```

```
## Error in eval(expr, envir, enclos): object 'my_Numbers' not found
```

Spelling error

```
my_numbers <- c(1:5)
my_numbers #correction
```

```
## [1] 1 2 3 4 5
```

```
#add two to vector named my_numbers
my_numbers <- c("1", "2", "3")
my_numbers + 2
```

```
## Error in my_numbers + 2: non-numeric argument to binary operator
```

Type error

```
my_numbers <- c("1", "2", "3")
as.numeric(my_numbers) + 2 #correction
```

```
## [1] 3 4 5
```

Syntax error, extra comma

```
#create a vector named my_numbers
my_numbers <- c(1, 2, 3, 4,)
```

```
## Error in c(1, 2, 3, 4, ): argument 5 is empty
```

```
my_numbers <- c(1, 2, 3, 4) #correction
```

Indexing error

```
#index the last element in my_numbers
my_numbers <- c(1, 2, 3)
my_numbers[4]
```

```
## [1] NA
```

```
my_numbers <- c(1, 2, 3)
my_numbers[3] #correction
```

```
## [1] 3
```

13.

$2 marks$

R comes with its own datasets, which you can access using command data() in the console. Take a look at the dataset, "HairEyeColor" and convert it into a tibble. Hint, if you have not already done so, install and load package "tibble".

```
#install.packages("tibble")
library(tibble)

data("HairEyeColor")
HairEyeColor <- as_tibble(HairEyeColor)
```

14.

$1 mark$

From HairEyeColor, index the "Eye" column.

```
HairEyeColor[["Eye"]]
```

```
##  [1] "Brown" "Brown" "Brown" "Brown" "Blue"  "Blue"  "Blue"  "Blue"  "Hazel"
## [10] "Hazel" "Hazel" "Hazel" "Green" "Green" "Green" "Green" "Brown" "Brown"
## [19] "Brown" "Brown" "Blue"  "Blue"  "Blue"  "Blue"  "Hazel" "Hazel" "Hazel"
## [28] "Hazel" "Green" "Green" "Green" "Green"
```

```
#alternatively
HairEyeColor$Eye
```

```
##  [1] "Brown" "Brown" "Brown" "Brown" "Blue"  "Blue"  "Blue"  "Blue"  "Hazel"
## [10] "Hazel" "Hazel" "Hazel" "Green" "Green" "Green" "Green" "Brown" "Brown"
## [19] "Brown" "Brown" "Blue"  "Blue"  "Blue"  "Blue"  "Hazel" "Hazel" "Hazel"
## [28] "Hazel" "Green" "Green" "Green" "Green"
```

15. $1 mark$

From HairEyeColor, sort by column "n"

```
arrange(HairEyeColor, n)
```

```
## # A tibble: 32 x 4
##    Hair  Eye   Sex        n
##    <chr> <chr> <chr>  <dbl>
##  1 Black Green Female     2
##  2 Blond Brown Male       3
##  3 Black Green Male       3
##  4 Blond Brown Female     4
##  5 Blond Hazel Male       5
##  6 Black Hazel Female     5
##  7 Blond Hazel Female     5
##  8 Red   Hazel Male       7
##  9 Red   Green Male       7
## 10 Red   Blue  Female     7
## # i 22 more rows
```

16. $1 mark$

From HairEyeColor, select only Hair, Eye and Sex columns (alternatively, you can remove column n).

```
 select(HairEyeColor, Hair,Eye,Sex)
```

```
## # A tibble: 32 x 3
##    Hair  Eye   Sex
##    <chr> <chr> <chr>
##  1 Black Brown Male
##  2 Brown Brown Male
##  3 Red   Brown Male
##  4 Blond Brown Male
##  5 Black Blue  Male
##  6 Brown Blue  Male
##  7 Red   Blue  Male
##  8 Blond Blue  Male
##  9 Black Hazel Male
## 10 Brown Hazel Male
## # i 22 more rows
```

```
#alternatively
select(HairEyeColor, -n)
```

```
## # A tibble: 32 x 3
##    Hair  Eye   Sex
##    <chr> <chr> <chr>
##  1 Black Brown Male
##  2 Brown Brown Male
##  3 Red   Brown Male
##  4 Blond Brown Male
##  5 Black Blue  Male
##  6 Brown Blue  Male
##  7 Red   Blue  Male
##  8 Blond Blue  Male
##  9 Black Hazel Male
## 10 Brown Hazel Male
## # i 22 more rows
```

17. $1 mark$

From HairEyeColor, filter the "Hair" column for Red only, save as "red_hair_data".

```
red_hair_data <- filter(HairEyeColor, Hair == "Red")
red_hair_data
```

```
## # A tibble: 8 x 4
##   Hair  Eye   Sex        n
##   <chr> <chr> <chr>  <dbl>
## 1 Red   Brown Male      10
## 2 Red   Blue  Male      10
## 3 Red   Hazel Male       7
## 4 Red   Green Male       7
## 5 Red   Brown Female    16
## 6 Red   Blue  Female     7
## 7 Red   Hazel Female     7
## 8 Red   Green Female     7
```

18. $1 mark$

From red_hair_data, recode the "Sex" column as Male = 1, Female = 2.

```
library(forcats)
red_hair_data <- red_hair_data %>%
  mutate(Sex = factor(Sex)) %>%
  mutate(Sex = fct_recode(Sex, "1" = "Male",
                          "2" = "Female"))

red_hair_data
```

```
## # A tibble: 8 x 4
##   Hair  Eye   Sex       n
##   <chr> <chr> <fct> <dbl>
## 1 Red   Brown 1        10
## 2 Red   Blue  1        10
## 3 Red   Hazel 1         7
```

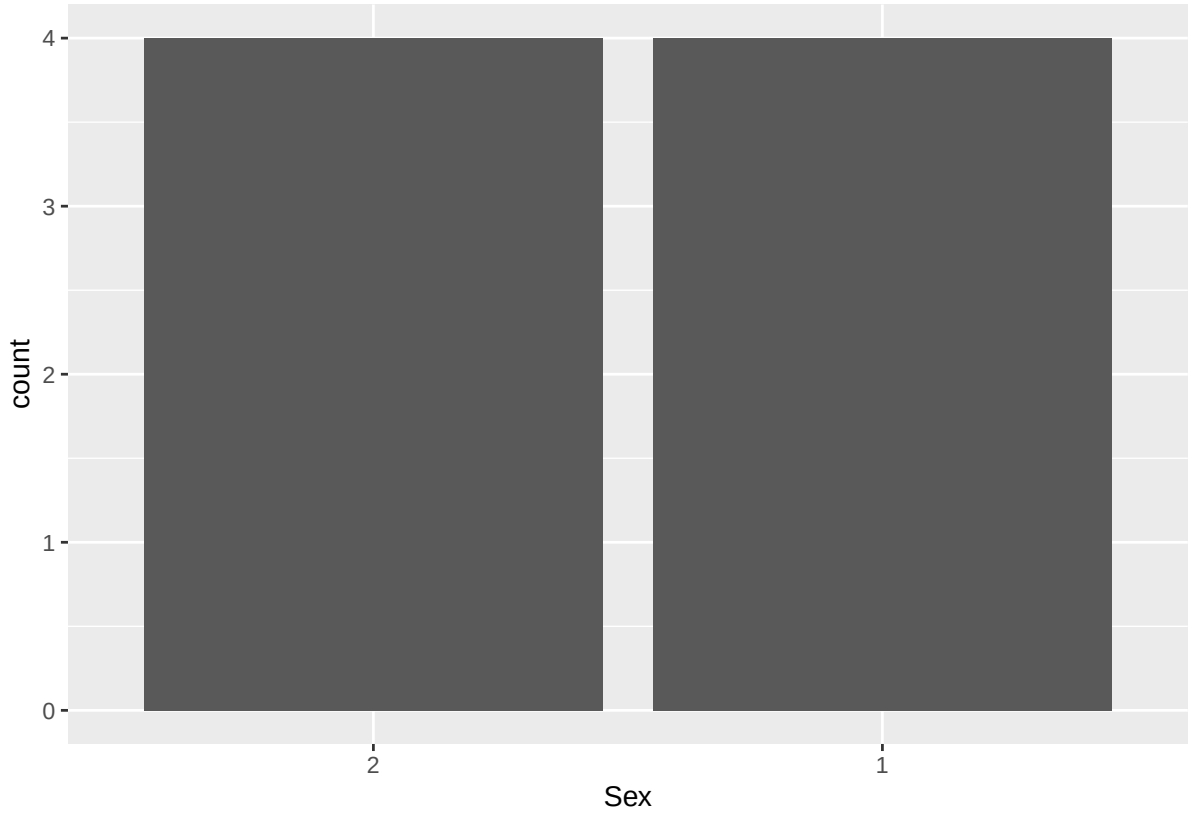
```
## 4 Red   Green 1         7
## 5 Red   Brown 2        16
## 6 Red   Blue  2         7
## 7 Red   Hazel 2         7
## 8 Red   Green 2         7
```

19.

$1 mark$

Plot the "Sex" column within red_hair_data. (Hint: use ggplot + geom_bar)

```
library(ggplot2)
red_hair_data %>%
  ggplot(aes(x = Sex)) +
  geom_bar()
```



Grading Scheme

The mark assigned for each question is indicated with the question. Use that to guide the answers. There will be additional marks assigned for submitting the Assessment in correct format.

Submission Instructions

$1 mark$

Remember: Submissions should only be in PDF format. When emailing the instructor, name PDF using format: LASTNAME_FirstInitial_A1.PDF.

E.g., GALLUCCI_J_A1.PDF.

$2 marks$

In your submission to the instructor, you must provide both the question number AND the question,in addition to your answers. Answer all the questions, in order.

How to Use R Markdown To Create A PDF With Answers

R Markdown is a formatting system that enables the creation of reproducible and dynamic reports using R. These reports allow for the inclusion of R code and its corresponding results in various formats such as slideshows, PDFs, HTML files, Word documents, and more. If you opt to use R Markdown to generate PDF files with solutions, it will require some time to familiarize yourself with R Markdown's syntax and capabilities. To simplify this process, a template named 'Assessment1_template.Rmd' has been provided, which you can utilize. To locate the provided R Markdown template, please navigate to the Assessments folder on GitHub: https://github.com/UofT-DSI/04-intro_r/tree/main/Assessments

Open this template file in RStudio

In order to generate PDF documents from R Markdown, two essential components are required:

- The "rmarkdown" R package
- The LaTeX distribution.

Note: There are various LaTeX options available, such as MiKTeX, MacTeX, TeX Live, and TinyTeX. You have the freedom to choose any of these LaTeX distributions that suits your needs. In this instance, I will demonstrate the use of TinyTeX. To install TinyTeX, you can utilize the "tinytex" R package.

```
# install the rmarkdown package
#install.packages("rmarkdown")
#library("rmarkdown")

# install the tinytex package
#install.packages("tinytex")
#library("tinytex")
# to install TinyTeX
#tinytex::install_tinytex()
```

The provided template already includes all the question numbers and the corresponding questions. You will only need to insert your answers. Before you begin entering your answers, please follow these steps:

1. Open the file 'Assessment1_template.Rmd' in RStudio.
2. Locate the 'Knit' icon in RStudio.
3. Click on the 'Knit' icon and select 'Knit to PDF' to generate a PDF output.

You are recommended to add small chunks of code at a time and 'Knit' the document. For more information including basics, see https://rmarkdown.rstudio.com/lesson-1.html or seek help during tutorial early.