STAT 33B Lab 1

Gunnar Mayer (3034535154)

R Markdown Documents

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can choose to knit your document to an HTML, Word, or PDF file.

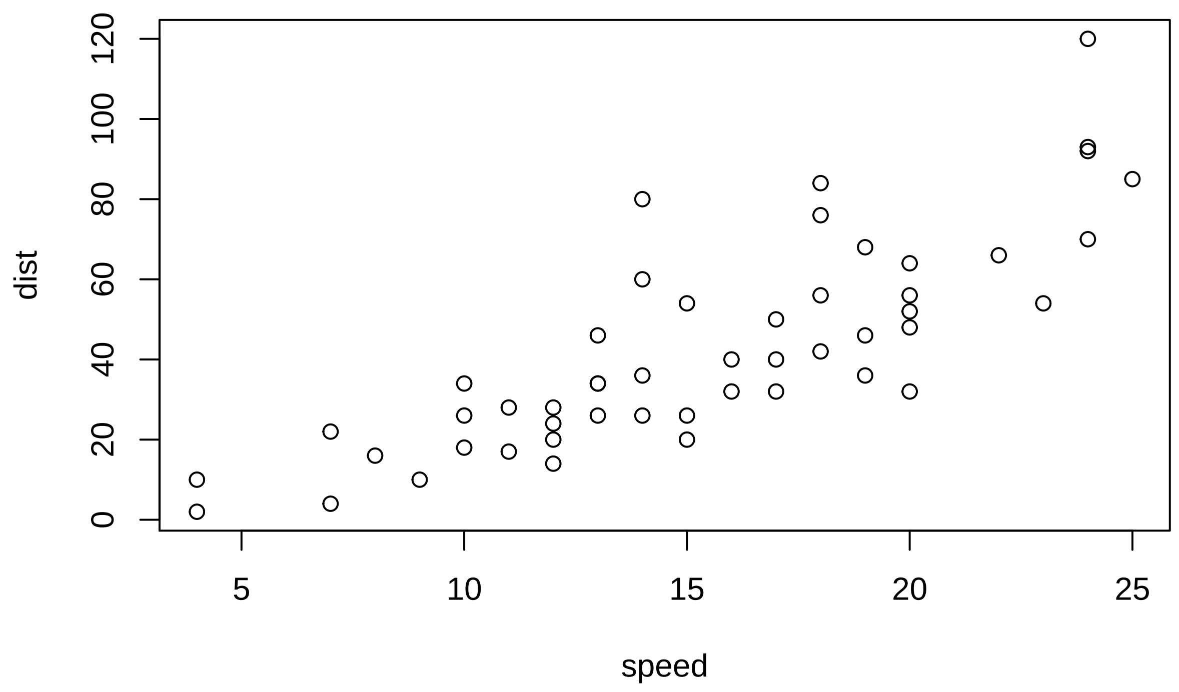
In order to knit to PDF, if you do not already have LaTex on your computer, you'll need to install the `tinytex` package in R. You can do this in R with the command `install.packages("tinytex")`. If you need help with this, come to office hours.

You can embed an R code chunk in the document like this:

```r
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00
```

You can also embed plots, for example:



Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

Submission Instructions

Edit this file, knit to PDF, and:

- Submit the Rmd file on bCourses.
- Submit the PDF file on Gradescope.

If you think you'll need help with submission, please ask in office hours *before* the assignment is due.

Answer all questions with complete sentences, and put code in code chunks. You can make as many new code chunks as you like. Please do not delete the exercises already in this notebook, because it may interfere with our grading tools.

As you work, you may find it helpful to be able to run your code. You can run a single line of code by pressing Ctrl + Enter. You can run an entire code chunk by clicking on the green arrow in the upper right corner of the code chunk.

Knit the document from time to time to make sure that your code runs without errors from top to bottom in a fresh R environment.

The code below controls the number of significant digits shown for the return values in your knitted document.

```r
options(digits = 3)
```

Exercise 1

Create a vector `y` with the values `3 5 9 17 33`. Use sequence generating functions such as `:` and vectorized math rather than `c()`.

```r
y = 2**(1:5)+1
y
```

```
## [1]  3  5  9 17 33
```

Exercise 2

Begin by loading the Best in Show "Dogs" data set (posted on bCourses; see lecture 2 for details). You'll need to edit the path to match where you downloaded the file on your computer.

```r
dogs = readRDS("/Users/g-mayer/adv_r/lab/lab_1/dogs_full.rds")
```

R provides a special value `NA`, called the missing value, as a placeholder for values that are unknown. The missing value can take the place of any of R's built-in types (integer, numeric, character, and so on).

Missing values are contagious: applying a binary operation to a missing value usually produces another missing value. For instance:

```r
5 + NA
```

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

Since we don't know the missing value, we can't possibly know its sum with another number, so the result is again a missing value. An upcoming lecture will provide more details about and examples of missing values.

You can use the `is.na` function to test whether a value is the missing value. Compute the number of missing values in the `longevity` column and assign it to the variable `num_na`.

```r
num_na = sum(is.na(dogs$longevity))
num_na
```

```
## [1] 37
```

Check your answer: There are about 30 to 40 missing values in the column.

Exercise 3

The values in the `weight` column are measured in pounds. Convert the entire column into kilograms (at Earth gravity) and assign the new measurements to `weight_kg`.

```r
weight_kg = dogs$weight/2.2046
head(weight_kg)
```

```
## [1]    NA  6.12 15.88  6.35    NA 13.61
```

Check your answer: If you did this right, the first few values should roughly be: `NA 6.12 15.88 6.35 NA 13.61`

Exercise 4

The values in the `height` column are measured in inches. Create a logical vector called `smaller_dogs` that has `TRUE` for dogs that are shorter than the median height (ignoring missing values) of the dogs.

Hint: read the documentation for the `median()` to find out how to ignore missing values.

```
smaller_dogs = dogs$height < median(dogs$height, na.rm = TRUE)
head(smaller_dogs)
```

```
## [1] FALSE    NA FALSE  TRUE  TRUE  TRUE
```

Check your answer: The first few values in the vector should be `FALSE NA FALSE TRUE TRUE TRUE`.

Exercise 5

The 5% trimmed mean is obtained by taking the mean of the middle 90% of the values (trimming off 10% of the values). Read the documentation for `mean()` and use this function to find the 5% trimmed mean of the `longevity` column. Assign the return value from the call to `mean()` to `longevity_mean5`

```
longevity_mean5 = mean(dogs$longevity, trim = 0.05, na.rm = TRUE)
longevity_mean5
```

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

Exercise 6

Create a vector called `weight_rnorm` that is the sum of the `weight` column and random values from a normal distribution with mean 0 and sd 2.

```
# Do not change the set.seed() line!
set.seed(37)
weight_rnorm = dogs$weight+ rnorm(length(dogs$weight), mean = 0, sd = 2)
head(weight_rnorm)
```

```
## [1]   NA 14.3 36.2 13.4   NA 29.3
```

Check your answer: The first few values of `weight_rnorm` should approximately be `NA 14.26 36.16 13.41 NA 29.33`.

Exercise 7

In this exercise, you'll investigate coercion. Run the following code and inspect the results, then write an explanation as to why each of the returned types makes sense.

```
3L + 3i
smaller_dogs * dogs$height
c(smaller_dogs, dogs$breed)
```

WRITE YOUR EXPLANATION BELOW:

"3L + 3i" returned 3+3i because the 'L' signifies an integer and the 'i' signifies a complex number. R interprets this as 3+3i, which is just adding a complex number to a simple integer.

The output returned by "smaller_dogs * dogs_height" makes sense because in R any number multiplied by the boolean value FALSE equates to 0, while any integer multiplied by TRUE returns that integer. Also, any number multipled by NA returns NA. And any number number multiplied by 0, is 0. This means that the only values we should see when smaller_dogs is multiplied by dogs_height is 0, some integer, or NA. Because smaller_dogs consists of TRUE / FALSE values and dogs_height contains NA, 0, some integer.

The vector returned by c(smaller_dogs, dogs$breed) makes sense because it is adding all the entries from smaller_dogs into a vector with all the entries from dogs_breed following afterwards. It converts everything to a string because vectors must be made up of the same data type. Since different breeds of animals cannot be translated into boolean values the vector converts everything into a string, except for NA values.

If dogs_breed had consisted of integers than when the two columns were vectorized together the boolean values in smaller_dogs would have turned into 1's and 0's

Exercise 8

For this exercise, you'll use the R package `lobstr` to inspect R's copy-on-write semantics. Install the `lobstr` package with the code:

```r
install.packages("lobstr")
```

For the rest of this exercise, you'll need to work in a regular R console rather than in RStudio (you should have both on your computer if you followed the setup instructions posted to the bCourse). In the R console, load the `lobstr` package with the code:

```r
library(lobstr)
```

The function `obj_addr()` in the `lobstr` package prints out the memory address of an R object. Here's some code that I ran in the R console earlier:

```r
x = 1:7 #  Won't cause a copy because it will create the original object
obj_addr(x)
# [1] "0x7fdf312e6690"

x[4] = 1000 # This will cause a copy because x is being changed
obj_addr(x)
# [1] "0x7fdf2d18b0a8"

x[5] = 2000 # This won't cause a copy because x has already been modified and copied
obj_addr(x)
# [1] "0x7fdf2d18b0a8"

y = x # This won't cause a copy because y just points to x's storage location
obj_addr(y)
# [1] "0x7fdf2d18b0a8"

y[3] = NA # This will cause a copy because y is being modified for the first time
obj_addr(y)
# [1] "0x7fdf2d0cf2c8"
```

```r
y[1] = 42 # This won't cause a copy because y has been modified
obj_addr(y)
# [1] "0x7fdf2d0cf2c8"

y = 1 + y # This will cause a copy because a copy of y is needed to create the new y
obj_addr(y)
# [1] "0x7fdf2d0cf418"
```

Run this code in your R console. You'll see different addresses, but should see the same pattern in when the addresses change.

Edit the code above to replace the memory addresses with the ones you found in your R console. Next to each assignment, add a comment that say whether the expression will cause a copy to be made in memory or not.

Are there any surprises? Can you figure out why?

EXPLAIN:

I was a little surprised by the last assignment, I guess I hadn't considered what happened at a fundamental level. I don't think the program could reassign y without having another copy to pull data from and add too. I experimented using 2 vectors and when you assign a vector to the value of another vector +1 it doesn't create a copy, unlike assigning a vector to itself +1.

Once you have completed these tasks, check that you have written your name at the top of this document and that the document can be knitted.

Finally, it's good practice to print the output of `sessionInfo()` so that others who want to reproduce your results know what packages and version of R you were using when you first did your analysis.

```r
# leave this as-is:
sessionInfo()
```

```
## R version 3.6.2 (2019-12-12)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Catalina 10.15.2
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] lobstr_1.1.1
##
## loaded via a namespace (and not attached):
##  [1] compiler_3.6.2  magrittr_1.5    tools_3.6.2     htmltools_0.4.0
##  [5] yaml_2.2.0      Rcpp_1.0.3      stringi_1.4.3   rmarkdown_2.1
##  [9] knitr_1.27      stringr_1.4.0   xfun_0.12       digest_0.6.23
## [13] rlang_0.4.2     evaluate_0.14
```