

Announcement

Question 1

Question 2

Question 3

Question 4

Question 5

Question 6

Homework 4

Neal Kar

ink2105

Announcement

Please do not add code folding (code_folding: hide) to your YAML Header or echo = FALSE to your RMD code chunk options. In order to accurately grade your HTML files we need to be able to see all of your code. Thank you!

Question 1

a)

Create a function that take a variable called `grade` which represents a score between 0 and 100 and outputs a letter grade. Here we define an `A` as 90 or above, `B` as 80 or above, `C` as 70 or above, `D` as 60 or above, and `F` as below 60. Test you function at least twice and show the results.

```
letter_grade <- function(grade) {  
  
    if(grade >= 90 & grade <= 100){  
        return("A")  
    }  
  
    if(grade >= 80 & grade < 90){  
        return("B")  
    }  
  
    if(grade >= 70 & grade < 80){  
        return("C")  
    }  
  
    if(grade >= 60 & grade < 70){  
        return("D")  
    }  
  
    if(grade >= 0 & grade < 60){
```

```
        return("F")  
    }  
}
```

```
letter_grade(95)
```

```
## [1] "A"
```

```
letter_grade(83.4)
```

```
## [1] "B"
```

```
letter_grade(71.345)
```

```
## [1] "C"
```

```
letter_grade(45)
```

```
## [1] "F"
```

b)

After a particularly rough semester, you are interested in finding the minimum grade you must receive on the final to pass class. The class points are broken down into 70% homework and 30% final exam.

Create a function that takes arguments `homework` which represents your homework average for the semester (0-100) and `final` which represents your desired score on the final (0-100) and returns a statement saying either "PASS" or "FAIL" using a C as a cutoff for passing.

```
pass_fail <- function(homework, final) {  
  
  hw_pts <- (0.7)*(homework)  
  
  final_exam_pts <- (0.3)*(final)  
  
  tot_pts <- hw_pts + final_exam_pts  
  
  if(tot_pts >= 70 & tot_pts <= 100) {  
    return("Pass")  
  }  
  
  if(tot_pts >= 0 & tot_pts < 70) {  
    return("Fail")  
  }  
  
}
```

c)

Assuming you averaged a 70 on the homework, what is the minimum final grade you must receive to pass the class? Use your function to guess and check and answer this question.

```
pass_fail(70, 70)
```

```
## [1] "Pass"
```

```
pass_fail(70, 60)
```

```
## [1] "Fail"
```

```
pass_fail(70, 65)
```

```
## [1] "Fail"
```

```
pass_fail(70, 68)
```

```
## [1] "Fail"
```

```
pass_fail(70, 69)
```

```
## [1] "Fail"
```

```
pass_fail(70, 69.99)
```

```
## [1] "Fail"
```

The minimum grade on the final exam to pass the class is a 70.

Question 2

a)

Create a function that takes argument `var` which represents a variable extracted from a tibble and returns a 95% confidence interval for the sample mean of the variable only if the variable is numeric. Otherwise, notify the function user that the entry is non-numeric.

```
CI_95 <- function(var) {  
  
  if(is.numeric(var)) {  
    mean_var <- mean(var)  
    t <- 1.960  
    sd_var <- sd(var)  
    n <- 60  
  
    lower_bound_95CI <- mean_var - (t*(sd_var/sqrt(n)))  
  
    upper_bound_95CI <- mean_var + (t*(sd_var/sqrt(n)))  
  
    CI_bounds <- c(lower_bound_95CI, upper_bound_95CI)  
  
    return(CI_bounds)  
  
  }  
  
  if(!is.numeric(var)) {
```

```
    return("The variable is not numeric. Cannot compute 95% C  
I.")  
  
}  
  
}
```

b)

Test this function on the supplement and length variables from the tooth growth dataset.

```
tooth_growth <- read_csv("data/ToothGrowth.csv")  
  
CI_95(tooth_growth$Supplement)
```

```
## [1] "The variable is not numeric. Cannot compute 95% CI."
```

```
CI_95(tooth_growth$Length)
```

```
## [1] 16.87779 20.74888
```

Question 3

a)

Many datasets are created with missing values labeled as 999 or 9999. Create a function that takes in a value and if it is 999 or 9999 changes it to NA (integer). Test your function on x, y, and z created below.

```
x = 999
y = 65
z = 9999

impute_miss_v1 <- function(value) {

  if(value == 999 | value == 9999) {
    value <- NA
    return(value)
  }

  if(value != 999 & value != 9999) {
    return(value)
  }

}
```

```
impute_miss_v1(x)
```

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

```
impute_miss_v1(y)
```

```
## [1] 65
```

```
impute_miss_v1(z)
```

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

b)

Similarly, some datasets will label missing values with two asterisks. Adapt your function so if the input is a character variable it will change all double asterisk signs to NA, and for numeric variables it changes 999 and 9999 to NA. Now, test the

function on w , x , y , and z as created below. Explain why it works as it does with each test.

```
w = "999"  
x = "char"  
y = "***"  
z = 88
```

```
impute_miss_v2 <- function(value) {  
  
  if(is.numeric(value) & (value == 999 | value == 9999)) {  
    value <- NA  
    return(value)  
  }  
  
  if(is.numeric(value) & (value != 999 & value != 9999)) {  
    return(value)  
  }  
  
  if(is.character(value) & value == "***") {  
    value <- "NA"  
    value <- as.numeric(value)  
  }  
}
```

```
    return(value)
}

if(is.character(value) & value != "***") {
    return(value)
}

}
```

```
impute_miss_v2(w)
```

```
## [1] "999"
```

```
impute_miss_v2(x)
```

```
## [1] "char"
```

```
impute_miss_v2(y)
```

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

```
impute_miss_v2(z)
```

```
## [1] 88
```

When value = w, the value satisfies the criteria of being a character but fails the test of being equal to "***". So, the function simply returns the original value.

When value = x, the value satisfies the criteria of being a character but fails the test of being equal to "***". So, the function simply returns the original value.

When value = y, the value satisfies the criteria of being a character and satisfies the test of being equal to "***". The value gets changed to "NA" (character) and then gets converted into numeric. So, the function returns NA (numeric).

When `value = z`, the value satisfies the criteria of being numeric but fails the test of being equal to 999 or 9999. So, the function simply returns the original value.

c)

Lastly, to improve your function, ensure it returns appropriate error messages. If the object given is a data frame, rather than an atomic object, have it inform the user (use `is.data.frame()` to test for whether the object is a data frame). Test your updated function with the objects `a`, `b`, and `c` created below.

```
a = 6
b = "red"
c = tibble(color = c("red", "green", "blue", "***"), number = c
(5, 10, 999, 12))
```

```
impute_miss_v3 <- function(value) {

  if(is.numeric(value) & (value == 999 | value == 9999)) {
    value <- NA
    return(value)
  }

  if(is.numeric(value) & (value != 999 & value != 9999)) {
    return(value)
  }

  if(is.character(value) & value == "***") {
    value <- "NA"
    value <- as.numeric(value)
  }
}
```

```
        return(value)
    }

    if(is.character(value) & value != "**") {
        return(value)
    }

    if(is.data.frame(value)) {
        return("You've entered a dataframe. Please enter an atomic o
bject.")
    }

}

impute_miss_v3(a)
```

```
## [1] 6
```

```
impute_miss_v3(b)
```

```
## [1] "red"
```

```
impute_miss_v3(c)
```

```
## [1] "You've entered a dataframe. Please enter an atomic object."
```

Question 4

a)

A collaborator has given you four data sets to analyze (the code below reads in the data). Unfortunately, all four are messy and need cleaning. Rather than clean each individually, write a function which cleans the dataset given as needed. Specifically, your function should:

(1) clean variable names

(2) remove the extra row

(3) remove the notes variable

(4) drop any rows with missing values

(5) remove entries with negative heights, lengths, or weights

(6) convert the height, weight, and length variables to numeric

(7) create a new variable called `new_var` which is the product of weight times height times length

```
table1 = read_excel("data/table1.xlsx")
table2 = read_excel("data/table2.xlsx")
table3 = read_excel("data/table3.xlsx")
table4 = read_excel("data/table4.xlsx")

clean_table <- function(df) {

  output <- df

  #Clean variable names
  output<- clean_names(output)

  #Remove extra row
  output <- filter(output, table_id != "extra")

  #Remove notes variable
  output<- select(output, -notes)

  #Remove rows with any missing values
```

```
output <- filter(output, !is.na(table_id))
output <- filter(output, !is.na(height_in_inches))
output <- filter(output, !is.na(weight_in_pounds))
output <- filter(output, !is.na(length))
output <- filter(output, !is.na(name))
```

#Remove entries with negative heights, lengths, or weights

```
output <- filter(output, height_in_inches >= 0)
output <- filter(output, weight_in_pounds >= 0)
output <- filter(output, length >= 0)
```

#Convert the height, weight, and length variables to numeric

```
output$height_in_inches <- as.numeric(output$height_in_inche
s)
output$weight_in_pounds <- as.numeric(output$weight_in_pound
s)
output$length <- as.numeric(output$length)
```

#Create a new variable called `new_var` which is the product

```
of weight times height times length
  output <- output %>%
    mutate(new_var = weight_in_pounds*height_in_inches*length)

  return(output)
}

table1_cleaned <- clean_table(table1)
table2_cleaned <- clean_table(table2)
table3_cleaned <- clean_table(table3)
table4_cleaned <- clean_table(table4)
```

b)

Combine all 4 resulting datasets (using `bind_rows`) and print the resulting data below using `kable`.

```
combined_table <- bind_rows(table1_cleaned, table2_cleaned, table3_cleaned, table4_cleaned)
```

```
kable(combined_table)
```

table_id	height_in_inches	weight_in_pounds	length	name	new_var
1	16	100	33	blue coffee table	52800
4	12	44	22	red table	11616
5	50	43	40	desk	86000
10	25	20	58	wood table	29000
17	18	30	80	dinning table	43200
18	22	15	80	kids table	26400

table_id	height_in_inches	weight_in_pounds	length	name	new_var
19	12	50	50	long table	30000
28	36	70	50	metal rim	126000
29	24	40	80	desk	76800

Question 5

Your colleague has shared a dataset for a medium sized depression study. This study had two active treatments and a control group over 10 weeks. The main outcome of interest was the PHQ-9 sum score, which can range from 0-27. The data is stored in the `depression_study.csv` file.

a)

During a meeting with your colleague they sketch out a graph they would like you to make for an upcoming poster session. Your colleague does not know exactly what the data will look like, but has a general idea of the graph they want. The picture they sketched, along with some notes, is included below in the `graph-sketch.jpg` file. Take a look at it by clicking on it in the file explorer in the lower right hand section of RStudio.

Important things to notice: (1) Your colleague is interested in mean PHQ-9 by treatment by day. (2) Your colleague wants the x and y axis to have specific ranges, tick marks, and labels. (3) Your colleague wants the colors of the three treatments to be different. (4) Your colleague wants the points that represent the three treatments to be different as well – squares, circles, and triangles. (5) Your colleague wants the legend below the plot.

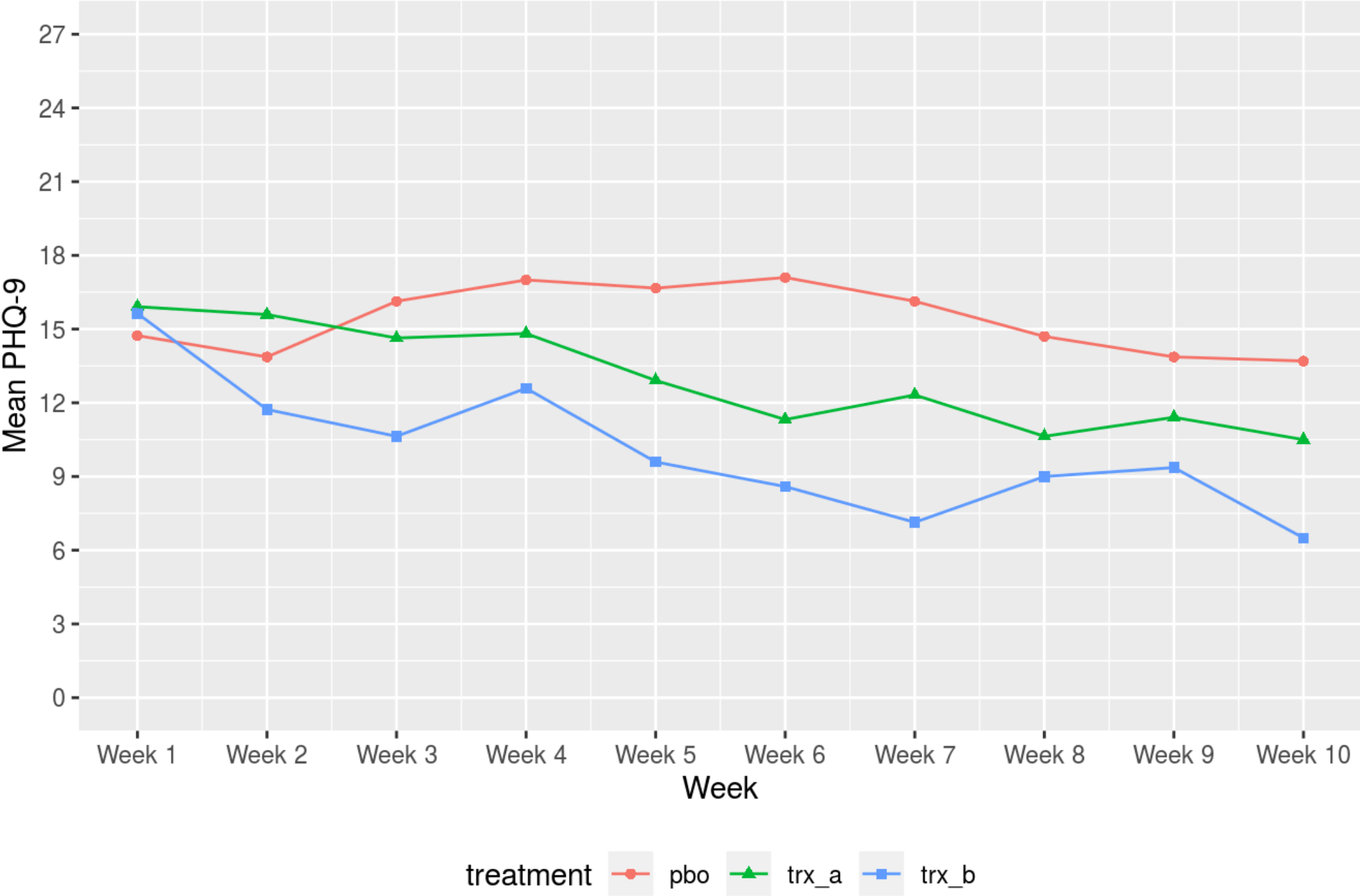
Use what you have learned about data manipulation and plotting to try to create a plot that matches their sketch to the best of your ability.

```
depression_data <- read_csv("data/depression_study.csv")

dep_data_weekly <- depression_data %>%
  group_by(treatment, week) %>%
  summarize(mean_phq9 = mean(phq9))

ggplot(data = dep_data_weekly) +
  geom_point(aes(x=week, y=mean_phq9, color=treatment, shape=treatment)) +
  geom_line(aes(x=week, y=mean_phq9, color=treatment)) +
  scale_x_continuous(breaks=1:10, labels = paste("Week", 1:10))
+
  scale_y_continuous(limits=c(0,27), breaks=seq(from=0,to=27,by=3)) +
  theme(legend.position = "bottom") +
  labs(title = "Observed PHQ-9 by Treatment Group", x = "Week",
y = "Mean PHQ-9")
```

Observed PHQ-9 by Treatment Group



Question 6

Make a list named `important_list` that contains the following VERY IMPORTANT entries:

`fav_book` Your favorite book. `age_hypothesis` The age when you first learned about hypothesis tests. `clt_fact` TRUE/FALSE: The distribution of the average of 30 observations or more will always be exactly normal. `fibonacci_start` A vector of the first seven numbers in the fibonacci sequence. `sample_df` A dataframe where the first column is five random normal samples ($\text{mean} = 0$, $\text{sd} = 1$), the second column is five random uniform samples ($\text{min} = 0$, $\text{max} = 5$), and the third column is five random poisson samples ($\text{lambda} = 2$).

Create this list and then use list indexing `[[]]` to report its elements using inline code in the sentences below, replacing the XXX's:

```
important_list <- list(fav_book="The Associate", age_hypothesis=
20, clt_fact=FALSE, fibonacci_start <- c(0,1,1,2,3,5,8), sample_
df <- tibble(normal_samples = rnorm(n = 5, mean = 0, sd = 1), un
iform_samples = runif(n = 5, min = 0, max = 5), poisson_samples
= rpois(n = 5, lambda = 2)))
```

My favorite book is The Associate. I first learned about hypothesis tests when I was 20 years old. It is FALSE that the distribution of the average of 30 observations or more will always be exactly normal. The fourth number of the fibonacci sequence is 2.