Assignment4

Yaniv Bronshtein

2/18/2021

Import the libraries

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.0.5     v dplyr   1.0.3
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.0

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(gapminder)
```

Question 1

Create a new function that, given an `lm` object, returns the top n residuals
arranged in descending order according to their largest absolute values
(but returns the residuals, not the absolute value of the residuals),
where the default value for n is 5. The function should give a clear error
message if n is larger than the number of residuals.
Demonstrate that your function works by applying it to mtcars.lm <- lm(mpg ~ disp, data = mtcars) first
with no argument for n, then with n = 6, and then with n = 40 (error message expected).

```
sort_res_by_abs <- function(obj, n = 5) {
  if (!("lm" %in% class(obj))) {
    cat("ERROR: obj must have class 'lm'", "\n")
    return(NA)
  }
  else {
    num_res <- length(obj$residuals)
    res <- as_tibble(obj$residuals)
    if (n > num_res) {
      cat("ERROR: the n value", n, "is larger than the number of residuals in the lm object",
```

```
        num_res, sep = " ","\n")
      return(NA)
    } else {
      sorted_res <- res %>% rename(res = value) %>%
      mutate(abs_res = abs(res)) %>%
      arrange(desc(abs_res)) %>%
      select(res) %>% head(n)
      cat("top", n, "residuals sorted by absolute value:\n", sep = " ")
      return(as.list(sorted_res))
    }
  }
}
```

Train the mtcars.lm object

```
mtcars.lm <- lm(mpg ~ disp, data = mtcars)
```

Demonstrate that the function works with no arguments

```
sort_res_by_abs(mtcars.lm)
```

```
## top 5 residuals sorted by absolute value:

## $res
## [1]  7.230540  6.086193  6.043775 -4.892201  4.719703
```

Demonstrate that the function works for n = 6

```
sort_res_by_abs(mtcars.lm, n = 6)
```

```
## top 6 residuals sorted by absolute value:

## $res
## [1]  7.230540  6.086193  6.043775 -4.892201  4.719703  3.937588
```

Demonstrate that the function displays an error message for n=40

```
sort_res_by_abs(mtcars.lm, n = 40)
```

```
## ERROR: the n value 40 is larger than the number of residuals in the lm object 32

## [1] NA
```

Demonstrate that the function displays an error message if an lm object is not provided

```
sort_res_by_abs("This is an object of the wrong class", n = 40)
```

```
## ERROR: obj must have class 'lm'

## [1] NA
```

Question 2

Read the file "height.txt" in the folder, and use regular expressions to clean the height variable and convert it into a numeric column representing height in inches. Determine and report the number of non-missing values of height for men and women. Finally, make a plot showing the distributions of height for men and for women on the same plot.

Read in height data and create a vector `heights_vec` for easier manipulation

```
height_t <- as_tibble(read_tsv("C:\\Users\\Julia\\Downloads\\height.txt")) %>%
  select(sex, height)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   time_stamp = col_datetime(format = ""),
##   sex = col_character(),
##   height = col_character()
## )
```

```
heights_vec <- height_t %>% pull(height)
```

This helper function is used to convert a height given in feet and inches to inches

```
convert_ft_2_inches <- function(feet, inches=0) {
  if (feet < 0 || feet >= 12 || inches < 0 || inches >= 12) {
    return(NA)
  }else {
    return(12 * feet + inches)
  }
}
```

These counters are strictly for debugging purposes

```
# Initialize counters
int_match_inches_count <- 0
float_match_inches_count <- 0
int_match_feet_count <- 0
float_match_feet_count <- 0
feet_ticks_match_count <- 0
feet_words_count <- 0
feet_words_count_decimal <- 0
feet_only_ticks <- 0
not_match_count <- 0
centimeters_count <- 0
```

Create a function to perform cleaning of height column in the tibble using regexes

```
# Thus function is used to perform cleaning of the height vector using regexes
# The input is a character vector and the output is a numeric vector
clean_height_data <- function(heights_vec) {
  # Create a numeric vector to store cleaned up height data
```

```r
cleaned_heights_vec <- numeric(0)
for (row in heights_vec) {
  row <- trimws(row)
  #In feet only with ticks
  if (str_detect(row, "^[1-9]'$")) {
    str_vec <- as.vector(str_extract_all(row, "[0-9.]+", simplify = TRUE))
    if (length(str_vec) ==1) {
      feet <- as.numeric(str_vec[1])
      result <- convert_ft_2_inches(feet)
      cleaned_heights_vec <- c(cleaned_heights_vec, result)
    }
  }
  # In inches whole number (1)
  else if (str_detect(row, "[1-9][0-9]{1,2}")) {
    cleaned_heights_vec <- c(cleaned_heights_vec, as.numeric(row))
    int_match_inches_count <- int_match_inches_count + 1
  }
  # In inches floating point number (2)
  else if (str_detect(row, "^[1-9][0-9]{1,2}\\.[0-9]{1,9}$")) {
    cleaned_heights_vec <- c(cleaned_heights_vec, as.numeric(row))
    float_match_inches_count <- float_match_inches_count + 1

  }

  # In feet whole number (3)
  else if (str_detect(row, "^[1-9]$")) {
    feet <- as.numeric(str_extract(row, "^[1-9]$"))
    cleaned_heights_vec <- c(cleaned_heights_vec, convert_ft_2_inches(feet, 0))
    int_match_feet_count <- int_match_feet_count + 1
  }
  # In feet floating point number(4)
  else if (str_detect(row, "^[1-9]{1}[\\.][0-9]{1,5}$")) {
    feet <- as.numeric(str_extract(row, "^[1-9].[1-9]{1,9}$"))
    cleaned_heights_vec <- c(cleaned_heights_vec, convert_ft_2_inches(feet, 0))
    float_match_feet_count <- float_match_feet_count + 1
  }
  #In inches only with ticks
  else if (str_detect(row, "^[0-9]{2,3}(\"|'')$")) {
    cleaned_heights_vec <- c(cleaned_heights_vec,
                             as.numeric(str_extract(row, "[0-9.]+")))

  }
  #In feet and inches with ticks(5)
  else if (str_detect(row, "^[1-9]{1}'([0-9]{1,2}(\\.[0-9]{1,9})?(\"|'')?)?$")) {
    feet_inches <- as.numeric(as.vector(str_extract_all(row,"[0-9.]+", simplify = TRUE)))
    cleaned_heights_vec <- c(cleaned_heights_vec,
                             convert_ft_2_inches(feet_inches[1], feet_inches[2]))
  }
  # In feet and inches with words(6)
  else if (str_detect(row, "^[1-9]{1}\\s*(ft | feet | foot)\\s*[0-9]{1,2}\\s*(in | inches)")) {
    feet_inches <- as.numeric(as.vector(str_extract_all(row,"[0-9.]+", simplify = TRUE)))
    cleaned_heights_vec <- c(cleaned_heights_vec,
                             convert_ft_2_inches(feet_inches[1], feet_inches[2]))
```

```
    }
    # In feet and inches with words and floating (7)
    else if (str_detect(row, "^[1-9]{1}\\s*(ft|feet|foot)\\s*[0-9]{1,2}\\.[0-9]{1,9}\\s*(in | inches)$")
      feet_inches <- as.numeric(as.vector(str_extract_all(row,"[0-9.]+", simplify = TRUE)))
      cleaned_heights_vec <- c(cleaned_heights_vec,
                               convert_ft_2_inches(feet_inches[1], feet_inches[2]))
    }
    # Height explicitly in centimeters
    else if (str_detect(row, "^[1-9][0-9]{2,3}\\s*(cm|centimeter|centimeters)$")) {
      cm_val <- as.numeric(str_extract(row, "[0-9.]+")) / 2.54
      cleaned_heights_vec <- c(cleaned_heights_vec, cm_val)
    }
    #No possible way to extract match. fill with NA
    else {
      cleaned_heights_vec <- c(cleaned_heights_vec, NA)
    }
  }
  return(cleaned_heights_vec)
}
```

Call the `clean_height_data()` function and create a modified tibble containing the new column

```
#Call clean_height_data to extract a cleaned numeric column
out <- clean_height_data(heights_vec)
```

```
## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion
```

```
## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion

## Warning in clean_height_data(heights_vec): NAs introduced by coercion
```

```
#Insert the cleaned column back into the height_t as a new column
height_t <- height_t %>% mutate(heights_cleaned=round(out)) %>%
  select(sex, heights_cleaned) %>% rename(height = heights_cleaned)
```

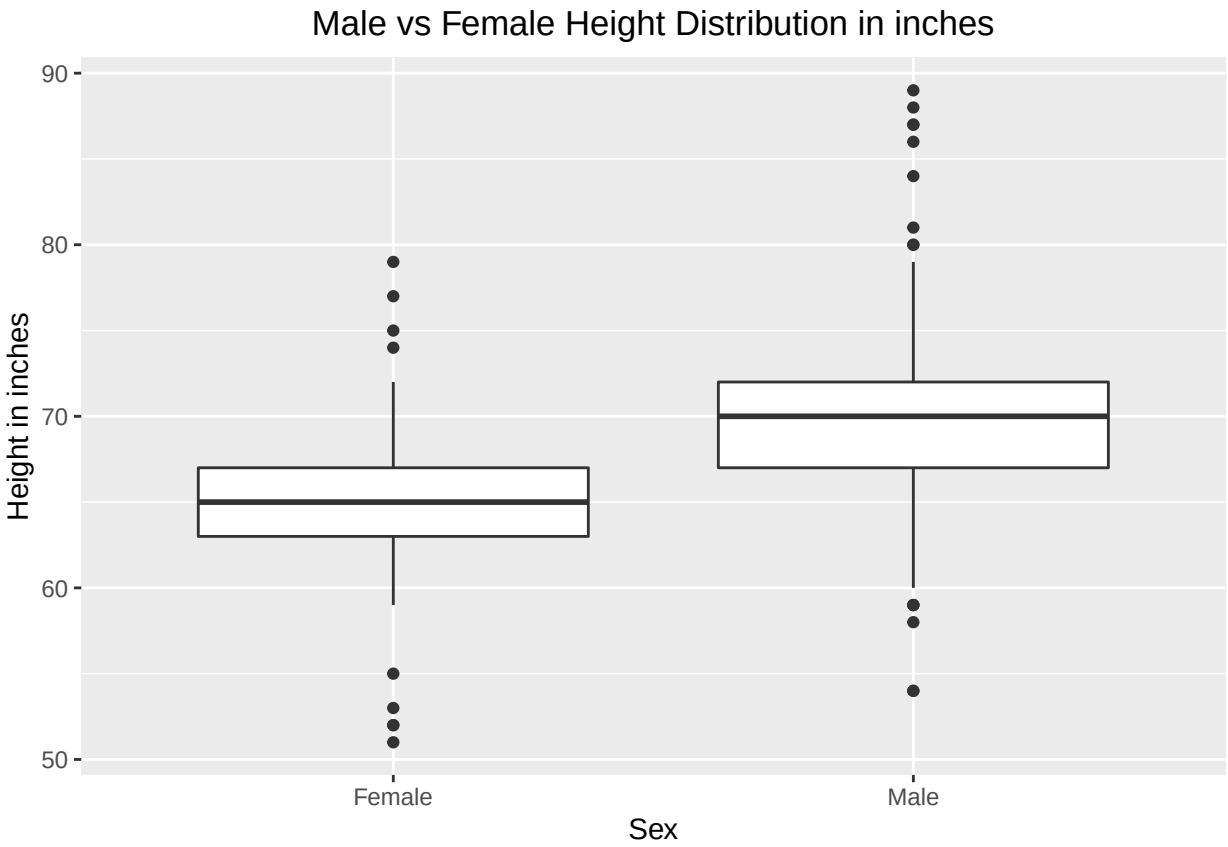
Derive the count of valid entries in the height column

```
count_na <- height_t %>% select(height) %>% is.na() %>% sum %>% as.numeric()
count_not_na <- dim(height_t)[1] - count_na
count_not_na
```

```
## [1] 1054
```

Create the ggplot but first filter the heights to remove outliers

```
height_t <- height_t %>% filter(height < 96, height > 50)
ggplot(mapping = aes(x = sex, y = as.numeric(height)), data = height_t) +
  geom_boxplot() +
  labs(
    title = "Male vs Female Height Distribution in inches",
    x = "Sex",
    y = "Height in inches"
  ) +
  theme(plot.title = element_text(hjust = 0.5))
```



Question 3

Split the gapminder data by country and use map() to calculate, by country, the R-squared for the linear model lifeExp ~ log10(gdpPercap). Using ggplot2, make a set of boxplots of R-squared by continent. **Load the gapminder data into a tibble**

```
gapminder_t <- as_tibble(gapminder)
```

Extract the values of R_squared for each country by performing the following steps: a). call split() to split the gapminder tibble data into separate tibbles by country b). Call map() to apply lm() which creates a linear model based on the equation lifeExp ~ log10(gdpPercap) Make sure to set the weights field to population **c).Use map yet again to call summary() on every lm() *object d).Now we call map_dbl() to create a list of all the R^2**

```
countries_rsquared <- gapminder_t %>%
  split(.$country) %>%
  map(~ lm(lifeExp ~ log10(gdpPercap), weights = pop, data = .)) %>%
  map(summary) %>%
  map_dbl("r.squared")
```

Now we need to extract all the country names of the countries and all the r^2 values to create a vertical tibble. We perform a left_join with the original gap_minder tibble to get all the original columns

```
continent_countries_t <- countries_rsquared %>%
  tibble(country = names(.), R_squared = .) %>%
  left_join(gapminder_t)
```

```
## Joining, by = "country"
```

For the tibble needed for the ggplot() we only need the columns continent and R_squared

```
final_t <- continent_countries_t %>% select(continent, R_squared)
```

Generate the ggplot

```
ggplot(data = final_t, mapping = aes(x = R_squared, y = continent)) +
  geom_boxplot()
```